CURSO PRÁTICO **8** DE PROGRAMAÇÃO DE COMPUTADORES

INPUT

PROGRAMAÇÃO BASIC - PROGRAMAÇÃO DE JOGOS - CÓDIGO DE MÁQUINA

NOVA CULTURAL

Cz$ 20,00

INPUT

Vol. 1 N.º 8

NESTE NÚMERO

FAÇA PROGRAMAS MAIS CURTOS

Economize espaço na memória de seu computador, aprendendo a criar programas mais curtos. Utilize comandos abreviados 141

ABAIXO DE ZERO

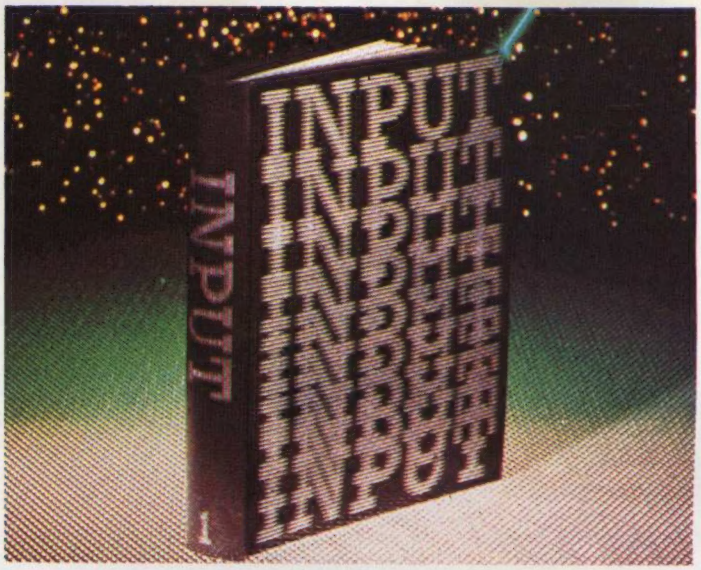
Números negativos: quando são necessários e como convertê-los aos sistemas binário e hexadecimal. Aprenda a "virar" os bits 142

ORDEM E LIMPEZA NO VÍDEO

Veja como organizar a disposição das informações no vídeo do computador 146

TORNE O JOGO MAIS DIFÍCIL

Como desenhar um labirinto aleatório. Duas maneiras de tornar mais difícil um jogo de labirinto. Aumente o placar, tirando "vidas" 153



PLANO DA OBRA

"INPUT" é uma obra editada em fascículos semanais, e cada conjunto de 15 fascículos compõe um volume. A capa para encadernação de cada volume estará à venda oportunamente.

COMPLETE SUA COLEÇÃO

Exemplares atrasados, até seis meses após o encerramento da coleção, poderão ser comprados, a preços atualizados, da seguinte forma: **1. Pessoalmente** — por meio de seu jornaleiro ou dirigindo-se ao distribuidor local, cujo endereço poderá ser facilmente conseguido junto a qualquer jornaleiro de sua cidade. Em São Paulo os endereços são: Rua Brigadeiro Tobias, 773 (Centro); Av. Industrial, 117 (Santo André); e, no Rio de Janeiro: Rua da Passagem, 93 (Botafogo). **2. Por carta** — Poderão ser solicitados exemplares atrasados também por carta, que deve ser enviada para DINAP — Distribuidor Nacional de Publicações — Números Atrasados — Estrada Velha de Osasco, 132 (Jardim Tereza) — CEP 06000 — Osasco — São Paulo. **3. Por telex** — Utilize o n.º (011) 33670 ABSA. Em **Portugal**, os pedidos devem ser feitos à Distribuidora Jardim de Publicações Ltd. — Qta. Pau Varais, Azinhaga de Fetais — 2685, Camarate — Lisboa; Tel. 257-2542 — Apartado 57 — Telex 43 069 JARLIS P.
Não envie pagamento antecipado. O atendimento será feito pelo reembolso postal e o pagamento, incluindo as despesas postais, deverá ser efetuado ao se retirar a encomenda na Agência do Correio. **Atenção**: Após seis meses do encerramento da coleção, os pedidos serão atendidos, dependendo da disponibilidade de estoque. **Obs.:** Quando pedir livros, mencione sempre o título e/ou o autor da obra, além do número da edição.

COLABORE CONOSCO

Encaminhe seus comentários, críticas, sugestões ou reclamações ao Serviço de Atendimento ao Leitor — Caixa Postal 9442, São Paulo — SP.

Editor
VICTOR CIVITA

REDAÇÃO

Diretora Editorial: Iara Rodrigues

Editor chefe: Paulo de Almeida
Editor de texto: Cláudio A.V. Cavalcanti
Editor de Arte: Eduardo Barreto
Chefe de Arte: Carlos Luiz Batista
Assistentes de Arte: Ailton Oliveira Lopes, Dilvacy M. Santos, José Maria de Oliveira, Grace A. Arruda, Monica Lenardon Corradi
Secretária de Redação/Coordenadora: Stefania Crema
Secretários de Redação: Beatriz Hagström, José Benedito de Oliveira Damião, Maria de Lourdes Carvalho, Marisa Soares de Andrade, Mauro de Queiroz
Secretário Gráfico: Antonio José Filho

COLABORADORES
Consultor Editorial Responsável: Dr. Renato M.E. Sabbatini (Diretor do Núcleo de Informática Biomédica da Universidade Estadual de Campinas)
Execução Editorial: DATAQUEST Assessoria em Informática Ltda. Campinas, SP.

Tradução: Maria Fernanda Sabbatini
Adaptação, programação e redação: Abílio Pedro Neto, Aluísio J. Dornellas de Barros, Marcelo R. Pires Therezo, Raul Neder Porrelli,
Coordenação geral: Rejane Felizatti Sabbatini
Assistente de Arte: Dagmar Bastos Sampaio

COMERCIAL
Diretor Comercial: Roberto Martins Silveira
Gerente Comercial: Flávio Ferrucio Maculan
Gerente de Circulação: Denise Maria Mozol

PRODUÇÃO
Gerente de Produção: João Stungis
Coordenador de Impressão: Atilio Roberto Bonon
Preparador de Texto/Coordenador: Eliel Silveira Cunha

Preparadores de Texto: Ana Maria Dilguerian, Antonio Francelino de Oliveira, Karina Ap. V. Grechi, Levon Yacubian, Maria Teresa Galluzzi, Paulo Felipe Mendrone
Revisor/Coordenador: José Maria de Assis
Revisoras: Conceição Aparecida Gabriel, Isabel Leite de Camargo, Ligia Aparecida Ricetto, Maria do Carmo Leme Monteiro, Maria Luiza Simões, Maria Teresa Martins Lopes.

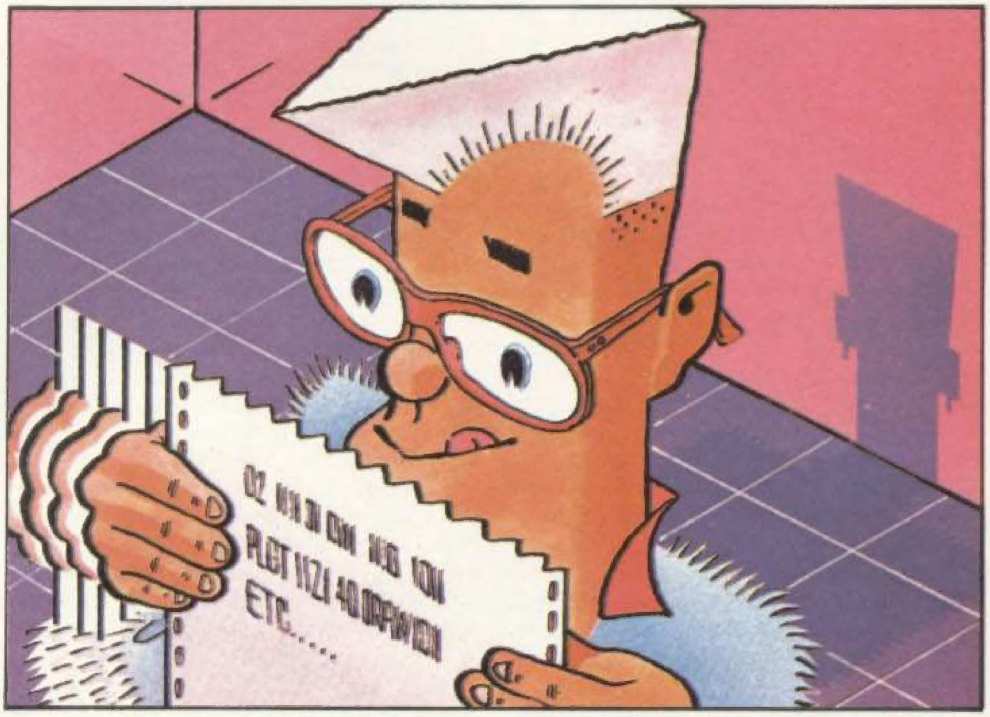
© Marshall Cavendish Limited, 1984/85.
© Editora Nova Cultural Ltda., São Paulo, Brasil, 1986.
Edição organizada pela Editora Nova Cultural Ltda.
(Artigo 15 da Lei 5 988, de 14/12/1973).

Esta obra foi composta na AM Produções Gráficas Ltda. e impressa na Divisão Gráfica da Editora Abril S.A.

FAÇA PROGRAMAS MAIS CURTOS

- USE COMANDOS ABREVIADOS
- COMO ECONOMIZAR MEMÓRIA
- COMANDOS MÚLTIPLOS
- COMO MELHORAR A VELOCIDADE DE UM PROGRAMA

Programas mais curtos ocupam menores espaços na memória do computador, além de serem de execução mais rápida. Aprenda a trabalhar com eles, melhorando o desempenho do seu micro.



Uma das regras básicas para se escrever um programa é torná-lo claro e legível, usando linhas curtas com comentários e espaços em branco e, se possível, com nomes longos de variáveis.

Evitar a digitação de muitos caracteres é uma das exigências mais importantes quando se pretende, por exemplo, copiar a longa listagem de uma revista. Neste caso, não há necessidade de digitar todas as declarações **REM** do programa. Afinal, sempre é possível consultar a revista mais uma vez, caso a finalidade de alguma seção do programa tenha sido esquecida.

Outra maneira de reduzir o trabalho de digitação na entrada de um programa é omitir palavras-chaves opcionais do BASIC, embora nem todos os micros admitam esse tipo de omissão (é o caso do Sinclair). Mas em outros computadores você pode deixar de lado todos os **LET** e **THEN**, que são declarações opcionais, ou mesmo — como no caso de uma linha de programa que contenha **IF...THEN GOTO...** — omitir o **THEN** ou o **GOTO** (a eliminação de ambas as partes, porém, não é possível).

Alguns micros permitem que se entrem palavras-chaves de forma abreviada, como ? em vez de **PRINT**, e ' (apóstrofo), para substituir **REM**, nos micros das linhas TRS-80 e TRS-Color.

Quando o programa for listado ou impresso, as abreviações aparecerão por extenso (com exceção do apóstrofo). Assim, embora isso poupe tempo de digitação, não leva a nenhuma economia de memória, e nem faz com que o seu programa rode mais rápido.

COMO ECONOMIZAR MEMÓRIA

Outra razão para se encurtar um programa é economizar espaço da memória. Essa economia é decisiva em programas muito longos.

Uma das formas de economizar bytes extras é utilizar nomes de variáveis com uma letra apenas. Esse procedimento poupa não só tempo de digitação, como também espaço de memória. Mas é sempre bom manter uma lista sobre o que faz cada variável, já que fica mais difícil usar códigos mnemônicos para variáveis com apenas uma letra.

Outra maneira de se encurtar um programa é omitir espaços em branco entre comandos, variáveis e operadores. Isso não se aplica ao ZX-81 e ao Spectrum (nesses micros, os espaços aparecem automaticamente). Em outros micros a omissão de espaços é uma medida extrema, pois ela torna muito difícil a leitura do programa. Uma regra, contudo, deve ser sempre observada: você deve deixar um espaço entre uma variável e o início de uma palavra-chave. Assim, uma linha como essa:

```
IF A = B AND B = C THEN PRINT "OK"
```

pode se tornar mais curta desse modo:

```
IFA=B ANDB=C THENPRINT"OK"
```

Mas, se você escreveu:

```
IF A=BAND B=C THENPRINT"OK"
```

o computador pensará que existe uma variável chamada **BAND**, e, como ela não existe, poderá ocorrer um erro durante a execução do programa (ou, em alguns micros, uma mensagem de erro).

Um terceiro modo de economizar memória consiste em combinar várias declarações em uma única linha. Na maioria dos computadores descritos aqui (linhas TRS-80, TRS-Color, MSX, Sinclair Spectrum, Apple II e TK-2000), o sinal de separação entre os comandos contidos em uma mesma linha é : (dois pontos). Apenas nos micros compatíveis com a linha ZX-81 isso não é possível. Mas esse recurso torna os programas mais difíceis de serem entendidos.

A maioria dos métodos aqui indicados tende a acelerar ligeiramente a velocidade de execução de um programa. Outro truque para se nuclear a execução de laços **FOR...NEXT** é omitir a variável que vem após o **NEXT** (isso não é permitido em alguns micros). Se o programa tiver vários laços aninhados, terminando na mesma linha, podemos colocar um único **NEXT**; por exemplo: **NEXT A, B, C** (apenas certos micros aceitam esta sintaxe).

ABAIXO DE ZERO

Diretamente relacionados com o funcionamento interno do computador, binários e hexadecimais apresentam problemas quando é preciso representar números negativos.

Durante a programação em código de máquina, é comum nos depararmos com números negativos, como por exemplo quando, numa programação de jogos, somos levados a movimentar figuras em determinadas direções ao longo da tela.

Como todos os outros, esses números devem ser codificados em bytes de oito bits, pois os computadores com unidades de memória desse tipo não contam com outros recursos para armazenagem de algarismos. Isto, porém, levanta um problema: como você já viu, um byte pode representar qualquer número de 0 a 255, ou 00000000 a 11111111, em binário. Mas isso esgota todas as possibilidades do binário de oito bits, já que não existe espaço para sinais de mais (+) ou menos (–), e nenhuma maneira pela qual eles possam ser representados nessa gama de valores. Em aritmética simples, se você subtrair 1 de 0 obterá –1. Agora, tente o mesmo cálculo em binário com oito bits:

```
00000000
       -1
--------
11111111
```

Esse resultado levará você, ao chegar ao oitavo bit à esquerda, a pedir "emprestado" um do próximo lugar à esquerda; mas quando o número binário é limitado a oito bits não existe nada à esquerda a ser emprestado. Do mesmo modo, se você subtrair outro 1 (para obter –2 em aritmética simples), obterá 11111110. Mas se 11111111 em binário equivale a 255 em decimal, então 11111110 deverá ser 254!

Imagine agora, por exemplo, o que poderia acontecer em decimal se você não tivesse mais que três dígitos ou "colunas" para colocar seus números. Veja o que acontece se você tentar acrescentar 999 a 100, quando tais limitações são impostas:

```
  100
+ 999
-----
(1)099
```

O número um, na unidade de milhar, teve que ser colocado entre parênteses. Em nossa aritmética de três dígitos, simplesmente não existe espaço para ele. Assim, o resultado dessa adição, em um sistema de três dígitos, será 99, que é exatamente o que você obterá se subtrair 1 de 100!

Do mesmo modo, em um sistema decimal de três dígitos, o resultado da soma de 998 mais 100 seria o mesmo da subtração de 100 menos 2. E subtrair 998 de 100 seria o mesmo que adicionar 2.

Recapitulando: a mesma série de algarismos em um byte de oito bits pode representar um número negativo e um positivo. Qualquer operação torna-se confusa e difícil diante disso.

O que você pode fazer em relação a isso? Bem, na maioria das aplicações em computadores domésticos, não há com que se preocupar: os endereços de memória e códigos de operação, ambos representados em binário, serão sempre considerados positivos. As únicas vezes em que você encontrará números negativos serão com dados ou com os chamados *saltos relativos,* os quais são códigos de máquina aproximadamente equivalentes às declarações **GOTO** do BASIC.

COMO "VIRAR" OS BITS

O processo utilizado em binário para se obter, a partir de um número positivo, o seu valor negativo, é conhecido como complemento de 2. Na realidade, essa técnica não tem uma base teórica muito bem fundamentada; o importante é que ela funciona.

Para se obter o valor negativo de um número binário, é necessário "virar" os bits e acrescentar 1. "Virar os bits" significa transformar os bits 1 em 0 e aqueles que têm valor 0, em 1.

No programa seguinte mostramos como isso funciona. Note que, quando o binário for limitado a oito dígitos, seu equivalente em hexadecimal preencherá exatamente dois dígitos. Isso significa que o hexadecimal equivalente ao complemento de 2 também atuará como o negativo.

```
10 CLS
20 PRINT@8,"NUMEROS NEGATIVOS";
30 PRINT@40,STRING$(16,CHR$(131
));
40 PRINT@65,"DEC"TAB(15)"BIN"TA
B(28)"HEX"
50 PRINT@226,"+"TAB(29)"+";
60 PRINT@361,"COMPLEMENTO DE 2"
70 PRINT@483,"0    0 0 0 0  0 0
 0 0     00";
80 FOR J=1474 TO 1502:POKE J,13
1:NEXT
90 FOR J=1 TO 7
100 FOR K=1 TO 24
110 POKE 1123+K+32*J,175
120 NEXT K,J
130 PRINT@174,"BITS";
140 PRINT@313,"+1";
150 AT=AT AND 255
160 T=AT: IF T=128 THEN SOUND 3
0,2
170 LN=3:GOSUB 280: GOSUB 310
180 T=T+256*(T>127):GOSUB 340
190 T=255-T:LN=7:GOSUB 280
200 T=T+1:T=T AND 255:LN=13:GOS
UB 280: GOSUB 310
210 T=T+256*(T>128):GOSUB 340
220 IN$=INKEY$:IF IN$<>"B" AND
IN$<>" " AND IN$<>CHR$(13) THEN
 220
230 IF IN$="B" THEN AT=AT-1:GOT
O 150
240 IF IN$=" " THEN AT=AT+1:GOT
O 150
250 PRINT @384,;:INPUT AT
260 PRINT @384,"     ";
270 GOTO 150
280 FOR X=0 TO 7
290 IF-(T AND 2^X) THEN PRINT @
LN*32+23-X*2+(X>3),"1";ELSE PRI
NT @LN*32+23-X*2+(X>3),"0";
300 NEXT:RETURN
310 IF T<16 THEN A$="0" ELSE A$
=""
320 PRINT @LN*32+29,A$+HEX$(T);
330 RETURN
340 PRINT @32*LN,MID$("   "+STR
$(T),LEN(STR$(T)));
350 RETURN
```

```
10 PRINT AT 0,7;"NUMEROS NEGA
TIVOS"
20 PRINT AT 2,1;"DEC";TAB 14;
"BIN";TAB 28;"HEX"
30 LET A$="
"
40 FOR N=7 TO 13
50 PRINT AT N,6; INVERSE 1;A$
60 NEXT N
70 PRINT AT 8,14;"BITS";
```

- QUANDO SÃO NECESSÁRIOS NÚMEROS NEGATIVOS
- A CONVERSÃO DE NÚMEROS NEGATIVOS
- A CONVENÇÃO DO SINAL

```
 BRIGHT 1;AT 12,21;"+1"
 80 PRINT AT 10,3;"+";AT 10,30
 ;"+"
 90 PRINT AT 15,7;"COMPLEMENTO
  DE 2"
 95 LET C=0
100 LET DD=-C: DIM A(8)
110 PRINT AT 4,0;"     ";AT 4,4
-LEN STR$ C;C
115 POKE 23608,C: LET E=PEEK
23608: LET Z=E: GOSUB 300:
PRINT AT 4,29;A$
120 PRINT AT 17,0;"     ";AT 17
,4-LEN STR$ DD;DD
130 LET D=128: LET CC=E
140 FOR N=1 TO 8: LET A(N)=0
150 IF CC-D>=0 THEN  LET A(N)=
1: LET CC=CC-D
160 PRINT  BRIGHT 1;AT 4,6+2*N
;A(N);AT 10,6+2*N;1-A(N)
170 LET D=D/2: NEXT N
180 POKE 23608,DD: LET DD=PEEK
23608: LET D=128
185 LET Z=DD: GOSUB 300: PRINT
AT 17,29;A$
190 FOR N=1 TO 8: LET B=0: IF
DD-D>=0 THEN  LET B=1: LET DD=
DD-D
200 PRINT  BRIGHT 1;AT 17,6+2*
N;B: LET D=D/2: NEXT N
210 PRINT AT 18,0;"----  -----
--------------  ----"
220 PRINT AT 19,3;"0    0 0 0
0 0 0 0 0       00"
230 IF INKEY$="" THEN  GOTO
 230
240 LET A$=INKEY$: IF A$=" "
THEN  LET C=C+1: IF C=128 THEN
 LET C=-128: SOUND 1,1
250 IF A$="B" OR A$="b" THEN
LET C=C-1: IF C=-129 THEN  LET
C=127: SOUND 1,1
260 IF A$<>" " AND A$<>"B" AND
A$<>"b" THEN  INPUT "?";C
270 GOTO 100
```

MICRO DICAS

COMO CONTAR EM COMPUTÊS

Se quisermos contar qualquer coisa em um computador, devemos começar sempre do zero e avançar daí para cima. Por exemplo, as locações de memória são numeradas em hexadecimal, de 0000 a FFFF.

O mesmo acontece com os bits em um byte. Quando numeramos os bits, começamos a partir da direita, com o bit número 0, e aumentamos a contagem à medida que nos deslocamos para a esquerda.

Assim, o dígito binário da extrema direita é chamado de bit 0. O que está imediatamente à sua esquerda é chamado de bit 1, o seguinte de bit 2 e assim por diante, até o bit 7 — aquele que está na extrema esquerda de um byte.

Assim como em política, os termos "à direita" e "à esquerda" podem, em certos casos, parecer um pouco confusos, especialmente quando lidamos com números de dois bytes que podem ser armazenados de cima para baixo, ou de baixo para cima, dependendo de como o computador opera.

Em "computês" correto (ou seja, na linguagem universal dos computadores), esses números são chamados de *valor mais significativo* — isto é, com o endereço mais alto — e *valor menos significativo*.

Dessa forma, se você estiver armazenando um endereço — 3D8E, por exemplo — 3D é o byte mais significativo, enquanto 8E é o byte menos significativo.

Os computadores das linhas Sinclair, TRS-80 e MSX colocam o byte menor ou menos significativo — 8E, no exemplo acima — na locação de memória mais baixa. O byte alto, que vale 100 em hexa (ou 256 em decimal), 3D neste caso, é colocado com o endereço mais alto na locação de memória (isto é, o endereço de memória que recebeu 8E, mais 1).

Existe uma única exceção para esta convenção de baixo/alto: os números da linha BASIC são armazenados em forma inversa.

O TRS-Color, por sua vez, armazena todos os números de dois bytes na locação de memória mais baixa com o byte alto, e armazena o byte baixo na locação de memória mais alta.

A mesma terminologia se aplica aos bits. O bit mais significativo no número binário 01010101, o bit 7, é 0, e o bit menos significativo, o bit 0, é o 1.

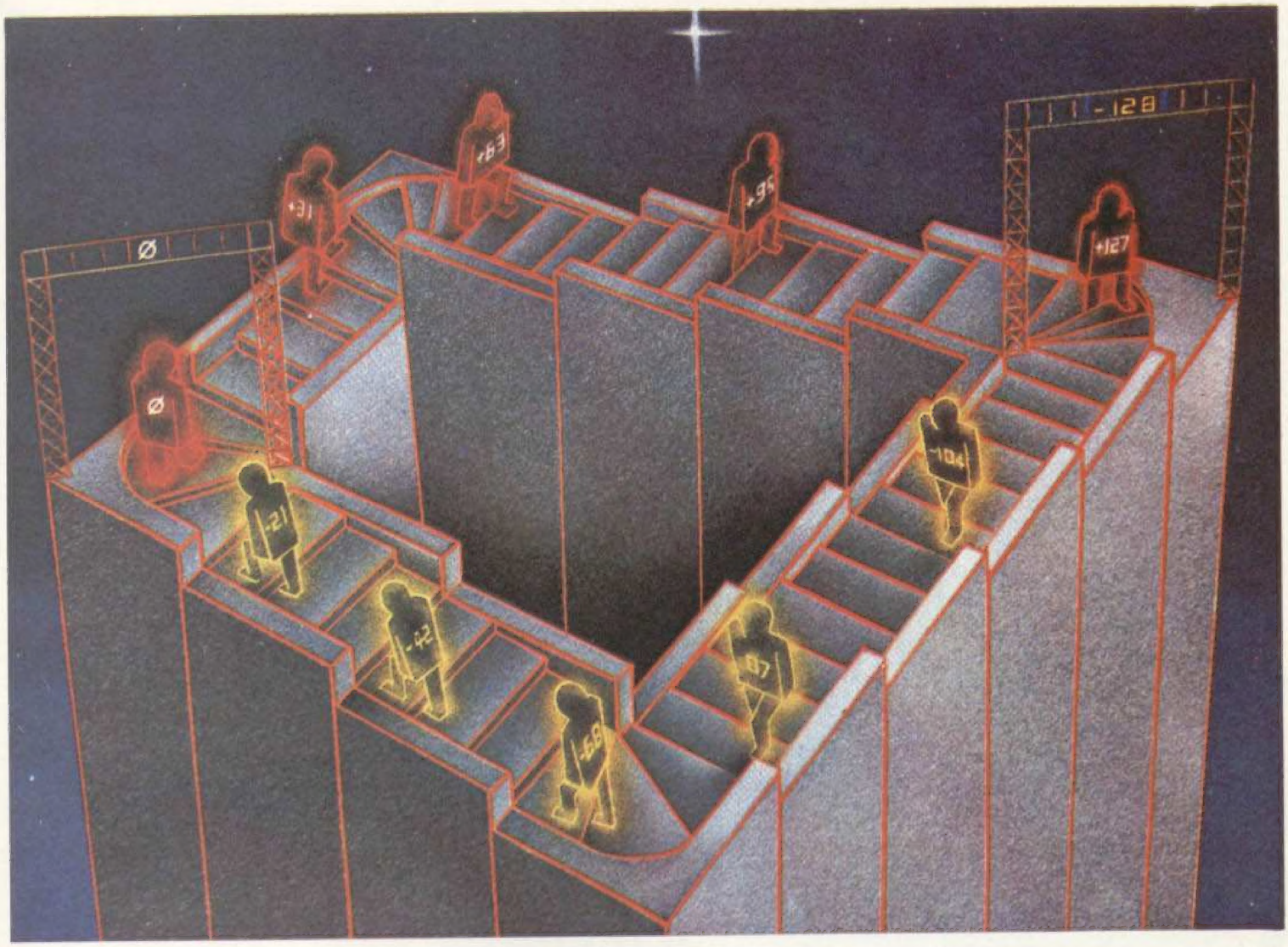
```
300 LET ZA=INT (Z/16): LET ZB=
Z-(16*ZA)
310 LET ZA=ZA+48: IF ZA>57
THEN  LET ZA=ZA+7
320 LET ZB=ZB+48: IF ZB>57
THEN  LET ZB=ZB+7
330 LET A$=CHR$ ZA: LET A$=A$+
CHR$ ZB: RETURN
```

SS

```
 10 PRINT AT 0,7;"NUMEROS NEGAT
 IVOS"
 20 PRINT AT 21,1;"DEC";TAB 14;
 "BIN";TAB 28;"HEX"
 30 LET A$="
 "
 40 FOR N=7 TO 13
 50 PRINT AT N,6;A$
 60 NEXT N
 70 PRINT AT 8,14;"BITS";AT 12,
 21;"+1"
 80 PRINT AT 10,3;"+";AT 10,30;
 "+"
 90 PRINT AT 15,8;"COMPLEMENTO
 DE 2"
 95 LET C=0
100 LET DD=-C
105 DIM A(8)
110 PRINT AT 4,0;"     ";AT 4,4-
LEN STR$ C;C
115 POKE 16507,C
116 LET E=PEEK 16507
117 LET Z=E
118 GOSUB 300
119 PRINT AT 4,29;A$
120 PRINT AT 17,0;"     ";AT 17,
4-LEN STR$ DD;DD
130 LET D=128
135 LET CC=E
140 FOR N=1 TO 8
145 LET A(N)=0
150 IF CC-D>=0 THEN LET A(N)=1
155 IF CC-D>=0 THEN LET CC=CC-D
160 PRINT AT 4,6+2*N;A(N);AT 10
,6+2*N;1-A(N)
170 LET D=D/2
175 NEXT N
180 POKE 16507,DD
181 LET DD=PEEK 16507
182 LET D=128
185 LET Z=DD
186 GOSUB 300
187 PRINT AT 17,29;A$
190 FOR N=1 TO 8
191 LET B=0
192 IF DD-D>=0 THEN LET B=1
193 IF DD-D>=0 THEN LET DD=DD-D
200 PRINT AT 17,6+2*N;B
205 LET D=D/2
207 NEXT N
210 PRINT AT 18,0;"----  ------
------------  ----"
220 PRINT AT 19,3;"0    0 0 0 0
0 0 0 0      00"
230 IF INKEY$="" THEN GOTO 230
240 LET A$=INKEY$
242 IF A$="F" THEN LET C=C+1
244 IF A$="F" AND C=128 THEN
LET C=128
250 IF A$="B" THEN LET C=C-1
255 IF A$="B" AND C=-129 THEN
LET C=127
260 IF A$<>"F" AND A$<>"B" THEN
INPUT C
270 GOTO 100
300 LET ZA=INT (Z/16)
305 LET ZB=Z-(16*ZA)
310 LET ZA=ZA+28
320 LET ZB=ZB+28
330 LET A$=CHR$ ZA
340 LET A$=A$+CHR$ ZB
350 RETURN
```

MSX

```
10 SCREEN 1:KEY OFF
14 VPOKE BASE(6)+27,166
20 LOCATE 6,0:PRINT "Números ne
gativos";
30 LOCATE 6,1:PRINT STRING$(17,
223);
40 LOCATE 3,3:PRINT "DEC";TAB(1
3);"BIN";TAB(24);"HEX";
50 LOCATE 1,10:PRINT "+";TAB(27
);"+";
60 LOCATE 6,17:PRINT "Complemen
to de 2";
70 LOCATE 0,21:PRINT "   0  0 0
 0 0  0 0 0 0  00 ";
80 FOR J=2 TO 30
84 VPOKE BASE(5)+J+19*32,223
88 NEXT
90 FOR J=1 TO 7
100 FOR K=1 TO 24
110 VPOKE BASE(5)+195+K+32*J,21
9
120 NEXT K,J
130 LOCATE 12,11:PRINT "BITS";
140 LOCATE 23,12:PRINT "+1"
150 AT=AT AND 255
160 T=AT
165 IF T=128 THEN SOUND 7,56:SO
UND 8,15:SOUND 1,1:FORI=1 TO 50
:NEXT:SOUND 8,0
170 LN=3:GOSUB 280
175 GOSUB 310
180 T=T+256*(T>127)
185 GOSUB 340
190 T=255-T:LN=7:GOSUB 280
200 T=T+1:T=T AND 255:LN=13
205 GOSUB 280:GOSUB 310
210 T=T+256*(T>128)
215 GOSUB 340
220 IN$=INKEY$:IF IN$<>"B" AND
IN$<>" " AND IN$<>CHR$(13) THEN
 220
230 IF IN$="B" THEN AT=AT-1:GOT
O 150
240 IF IN$=" " THEN AT=AT+1:GOT
O 150
250 LOCATE 0,20:INPUT AT
260 LOCATE 0,20:PRINT "     ";
270 GOTO 150
280 FOR X=0 TO 7
290 IF -(T AND 2^X) THEN VPOKE
BASE(5)+LN*32+87-X*2+(X>3),ASC(
"1") ELSE VPOKE BASE(5)+LN*32+8
7-X*2+(X>3),ASC("0")
300 NEXT:RETURN
310 IF T<16 THEN A$="0" ELSE A$
=""
320 LOCATE 25,LN+2:PRINT A$+HEX
$(T);
```




```
330 RETURN
340 LOCATE 0,LN+2:PRINT MID$("
  "+STR$(T),LEN(STR$(T)));
350 RETURN
```

Na convenção de sinal, o número 128 (ou 10000000, em binário) atua como seu próprio negativo e o computador recomeça a contar de forma ascendente.

Um número binário ou hexadecimal pode, na maior parte das vezes, atuar perfeitamente bem, seja ele um número positivo ou negativo. Mas, às vezes, você precisa saber se um número é positivo ou negativo.

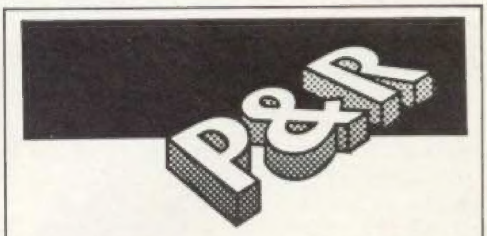
Quando se quer que um programa dê um salto, em código de máquina, é preciso especificar quantos bytes deve ter o salto a ser dado pelo computador: com um número positivo, no caso do salto para a frente; com um número negativo, para saltar para trás.

O que o computador faz é olhar para o primeiro bit do número binário e decidir se o número é negativo ou positivo. Se o primeiro bit for 1, o computador o tomará como negativo. Se for 0, ele será considerado como positivo.

Conhecido como *convenção de sinal* esse processo indica que, ao invés de tomar números binários de oito bits para representar a série de 0 a 255, o computador a trata como −128 a +127.

No programa de conversão de complemento de 2, você notará que o computador se atrapalha quando chega ao 128. Isso se deve a que o 128, ou seja, 10000000 em binário, ou 80 em hexa, atua como o seu próprio negativo (você pode verificar por si mesmo: −10000000 + 10000000 = (1)00000000 ou 0 em binário de oito bits, do mesmo modo que 80 + 80 = (1)00 ou 0 em hexa de dois dígitos. E 128 − 128 = 0 em decimal).

Portanto, qual deles é positivo ou negativo? O 10000000 tem o 1 em seu primeiro bit; o computador o trata como um número negativo: − 128. Por outro lado, o 0 — ou 00000000 — tem o 0 em seu primeiro bit; desse modo, o computador o trata como positivo.



Quando posso utilizar o sistema decimal codificado em binário (BCD)?

O BCD é empregado para imprimir números na tela, ou transmiti-los rapidamente. Para codificar um decimal de dois dígitos em um byte em binário, colocamos o dígito à direita nos últimos quatro bits significativos (um " meio byte") e o dígito à esquerda dentro do meio byte mais significativo.

Existem dezesseis dígitos que podem ser codificados em quatro bits binários. O BCD é um número hexadecimal sem as letras A, B, C, D, E e F. O único problema surge quando obtemos um número maior que 9 em um dos meios bytes.

ORDEM E LIMPEZA NO VÍDEO

- COMO UTILIZAR **TAB** COM **PRINT**
- SINAIS DE PONTUAÇÃO
- COMO MELHORAR A EXIBIÇÃO EM TELA

Listas, tabelas e instruções são mais compreensíveis quando distribuídas pelo vídeo de forma clara e ordenada. Use comandos do BASIC para isso.

Existem muitas maneiras de se dispor textos na tela. Mas é importante organizar bem a exibição, especialmente se você está escrevendo um programa que outras pessoas utilizarão.

Cada computador possui uma maneira própria de posicionar o texto na tela. Algumas funções de posicionamento são padronizadas para todas as variantes do BASIC, independentemente da marca do computador (exemplo: a função **TAB** e os sinais de pontuação — vírgulas e pontos e vírgulas — utilizados para separar os elementos de uma declaração **PRINT**). Entretanto, há outros comandos (associados ou não ao **PRINT**) destinados a posicionar o cursor na tela por meio do endereçamento por linhas e colunas e que variam amplamente entre os diversos computadores:

Comando	Microcomputador
PRINT AT	Sinclair
PRINT @	TRS
LOCATE	MSX
HTAB e **VTAB**	Apple

```
PRINT "BOM DIA"
```

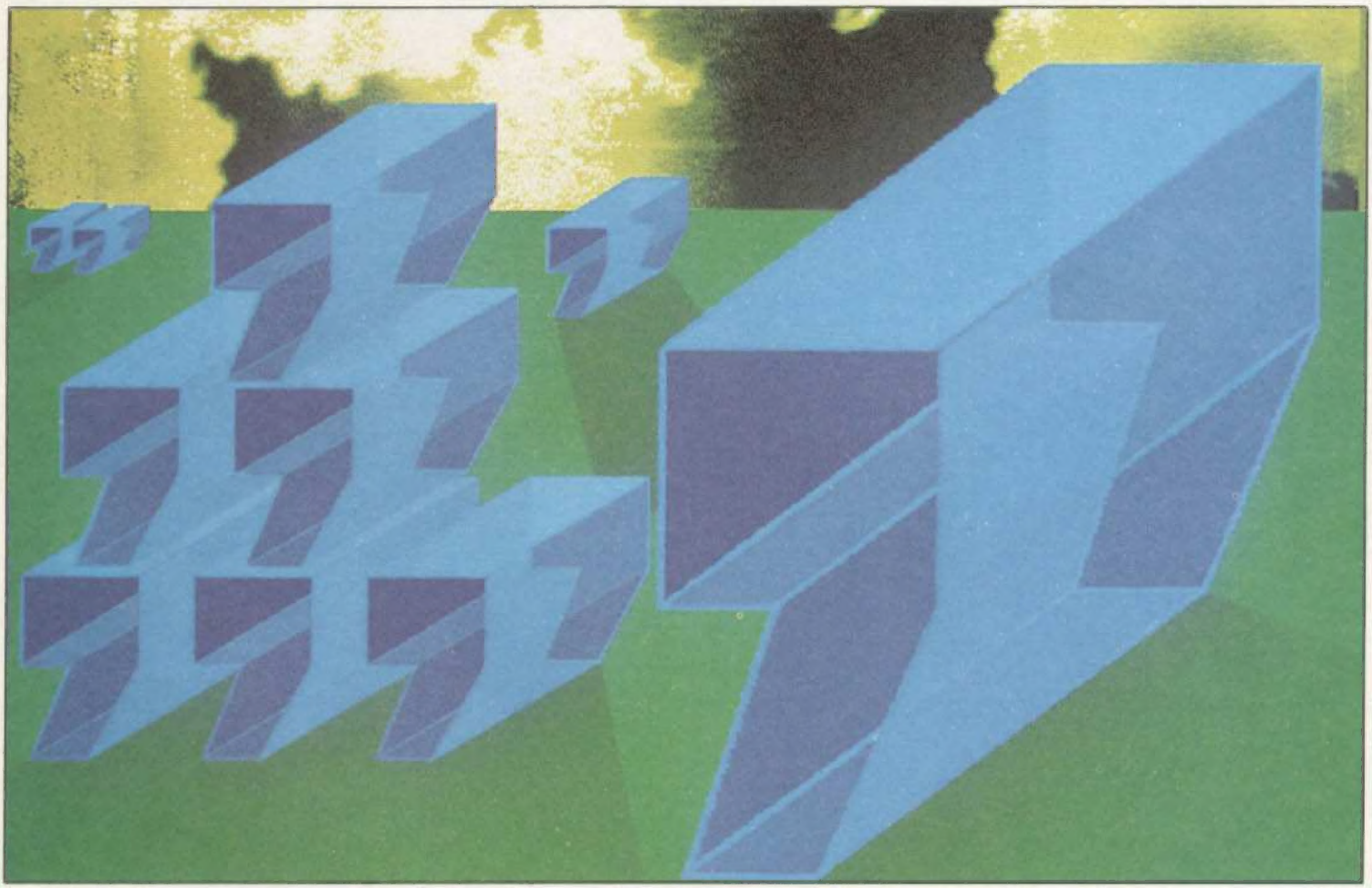
imprimirá a mensagem na próxima linha em branco, a partir da margem esquerda da tela.

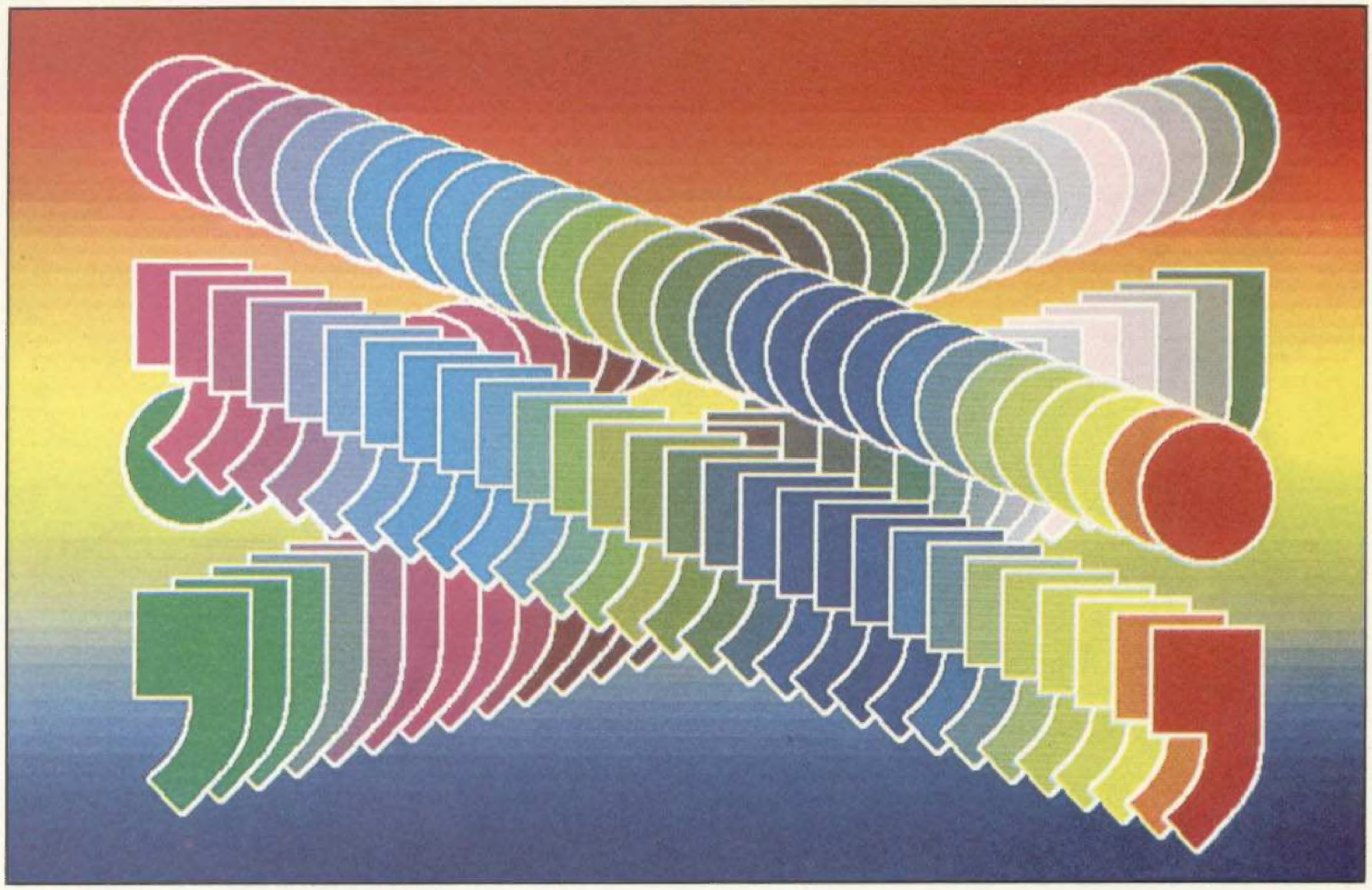
Entretanto, existem meios para dizer ao computador como imprimir a mensagem em uma posição diferente:

```
PRINT TAB(20)"BOA TARDE"
```

A função **TAB** é usada em conjunto com um **PRINT**. O número entre parênteses especifica a coluna a partir da qual o texto (ou número) que se segue será impresso. Pode-se colocar também uma expressão numérica ou uma variável.

O resultado da expressão numérica ou o número colocado como parâmetro podem ser fracionários, mas o computador usará apenas a sua parte inteira (por exemplo, **TAB (23, 2197)** será aproximado para **23**). O número resultante deve ficar entre 0 e o número máximo de colunas por linha do computador. É possível também posicionar cada palavra separadamente. Modifique a linha para:





```
PRINT TAB(20)"BOA" TAB(30)"
TARDE"
```

A primeira palavra é impressa a partir da coluna 20, e a segunda, a partir da coluna 30 (**TAB** negativo não funciona, ou seja, não se pode recuar posições na linha de impressão).

Não existe espaço em branco entre a palavra-chave **TAB** e o primeiro parêntese. Alguns micros exigem ponto e vírgula entre o **TAB** e o elemento que o segue:

```
PRINT TAB(20);"BOA";
TAB(30);"TARDE"
```

SINAIS DE PONTUAÇÃO

A função **TAB** indica a coluna, numa linha de impressão na tela, a partir da qual o texto será exibido.

O comando **PRINT** emprega vírgula e ponto e vírgula (e também apóstrofo, na linha Sinclair Spectrum), para determinar o espaço entre os elementos a serem impressos.

Quando dois elementos de uma linha **PRINT** são separados por um ponto e vírgula, o computador não concede espaço entre eles e:

```
PRINT "TOCA";"DISCOS"
```

imprimirá TOCADISCOS, sem separação. Se os elementos de **PRINT** forem números, e não cordões alfanuméricos, o resultado dependerá do tipo de micro. Alguns colocarão um espaço em branco antes do número a ser impresso. Assim,

```
PRINT "SOMA";25
```

mostrará o resultado: SOMA 25.

No Sinclair, entretanto, o resultado dessa instrução seria: SOMA25.

A vírgula funciona como a função **TAB**. A cada vírgula encontrada, o programa coloca a próxima informação a ser impressa numa posição distante dez colunas (ou dezesseis, no Sinclair) à direita da última posição inicial. Assim,

```
PRINT "TOCA","DISCOS"
```

produzirá:

TOCA DISCOS.

Em muitos casos, especialmente quando há colunas de números ou de palavras, a informação pode ser posicionada muito mais facilmente e com muito menos esforço de sua parte, exigindo o emprego da função **TAB** apenas de vez em quando. Digite e execute as linhas a seguir, para ter uma idéia de como isso funciona:

```
10 PRINT "012345678901234567890
1234567890"
20 PRINT 9;9
30 PRINT 9,9
```

A linha 10 numera as colunas no topo da tela. Uma vírgula separa os números em "campos", cada um com dez colunas de extensão (dezesseis, no Sinclair). Agora tente o mesmo programa com textos (isto é, cordões alfanuméricos), acrescentando essas linhas:

```
40 LET A$="A"
50 PRINT A$;A$
60 PRINT A$,A$
```

Normalmente, costumamos alinhar palavras à esquerda e números à direita, mas os computadores descritos aqui não fazem isto automaticamente.

O ponto e vírgula é essencial para que a função **TAB** funcione corretamente, pois uma vírgula após um **TAB** provocará o deslocamento para uma nova posição de tabulação. O ponto e vírgula também é útil no final de uma declaração **PRINT**. Sua função, neste caso, é manter o cursor de impressão na tela, na última posição impressa, de tal forma que o próximo comando **PRINT** encontrado pelo programa comece a imprimir nessa posição, sem mudar de li-

nha na tela. O programa a seguir imprime todo o alfabeto utilizando um laço **FOR...NEXT** para incrementar de um em um os códigos ASCII para os caracteres de A a Z.

```
20 FOR C=65 TO 90
30 PRINT CHR$(C);
40 NEXT C
```

(Para micros da linha Sinclair ZX-81, substitua a linha 20 por **FOR C = 38 TO 63**, pois eles usam um código diferente do ASCII.) Se você não colocar o ponto e vírgula ao final da linha 30, cada letra será impressa em uma linha separada. Os sinais de pontuação e o **TAB** oferecem tantas maneiras de se exibir textos na tela que você encontrará certamente combinações adequadas aos seus próprios programas.

SS

As linhas Sinclair contam com três declarações para posicionamento de textos na tela: **PRINT, PRINT AT** e **PRINT TAB**. A declaração **PRINT** sozinha imprime uma linha em branco (veja mais adiante).

A tela do Sinclair tem 22 linhas, numeradas de 0 a 21, a partir do topo da tela. Cada linha, por sua vez, tem 32 "colunas", ou posições, para os caracteres, numerados de 0 a 31, a partir da margem esquerda da tela. O comando **PRINT AT** indica onde o texto a ser impresso vai começar: os números (ou expressões) que seguem **AT** indicam a linha e a coluna; as linhas:

```
PRINT AT 0,0,"*"
PRINT AT 21,0,"*"
PRINT AT 0,31,"*"
PRINT AT 21,31,"*"
```

imprimirão um asterisco em cada canto do vídeo. Se você quiser imprimir mais de um caractere em uma locação da tela, o computador imprimirá o primeiro caractere onde você determinar e o restante nas demais posições da mesma linha.

```
PRINT AT 10,12;"COMPRIMENTO"
```

coloca o "C" na linha 10, coluna 12, e as outras letras em 10,13 , 10,14.... etc., até 10,17.

Outro recurso interessante do **PRINT AT** é que uma linha **PRINT** pode conter vários **AT**, separados por pontos e vírgulas. Por exemplo, a linha usada para imprimir quatro asteriscos poderia ser assim:

```
PRINT AT 0,0,"*"; AT 21,0,"*";
    AT 0,31,"*"; AT 21,31,"*"
```

PRINT TAB

No Sinclair, a instrução **PRINT TAB** funciona, em muitos casos, do mesmo modo que **PRINT AT**. Contudo, existem duas diferenças: não é preciso especificar a linha; e não se pode utilizar o **PRINT TAB** para escrever (nem para apagar) por cima de qualquer material na tela. Se você entrar um novo **PRINT TAB** na antiga posição, a impressão será realizada em uma linha abaixo da anterior. Agora, digite:

```
PRINT AT 0,15;"*"
PRINT AT 0,15;"?"
```

Limpe a tela (**CLS**) e tente isso:

```
PRINT TAB 15;"*";
PRINT TAB 15;"?";
```

Comparado com o **PRINT AT**, o **PRINT TAB** economiza trabalho de digitação e isenta o operador da tarefa de lembrar-se dos números de linhas à medida que vai prosseguindo.

O programa a seguir permite que você digite as notas dos alunos de uma classe e as disponha ordenadamente na tela, usando uma série de funções **TAB**; além disso, ele calcula a média das notas (digite tudo em letras maiúsculas, para micros da linha ZX-81):

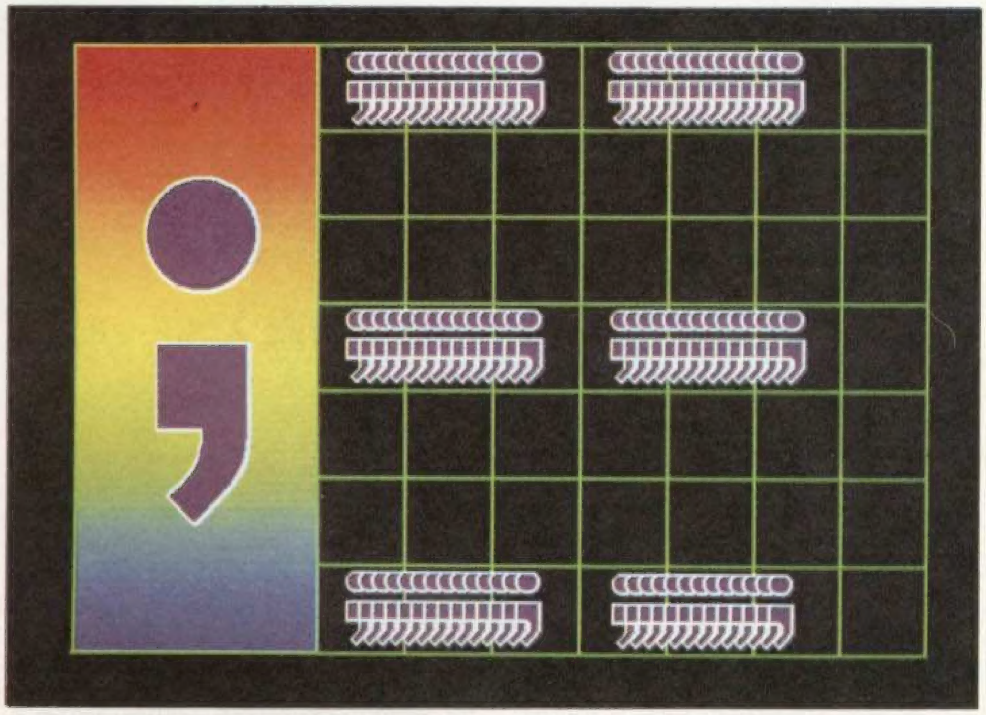
```
10 PRINT "NOME";TAB 12;"PROVA"
;TAB 18;"ORAL";TAB 23;"EXAME";
TAB 29;"MED"
20 PRINT AT 20,0;"Nome ?"
30 INPUT n$
35 PRINT AT 20,0;"Prova?"
40 INPUT np
45 PRINT AT 20,0;"Oral ?"
50 INPUT no
55 PRINT AT 20,0;"Exame?"
60 INPUT ne
70 LET m=((np+no*2)/3+ne)/2
80 PRINT N$;TAB 14;np; TAB 19;
no; TAB 24;ne; TAB 29; m
90 PRINT
100 PAUSE 200
110 GOTO 20
```

As linhas **PRINT**, usadas para pedir os dados iniciais de cada aluno, são posicionadas com o **AT** na penúltima linha da tela.

Ao ser rodado pela primeira vez, o programa limpa a tela e coloca na primeira linha as identificações correspondentes às colunas de dados. Em seguida ele perguntará: "NOME?". Digite o nome do aluno. Depois, perguntará sucessivamente as notas da prova, do exame oral e do exame final. Entre números inteiros ou, no máximo, com uma decimal. Em seguida o programa mostrará esses dados de entrada, juntamente com a média. Adicione as linhas:

```
75 LET M=INT(M*10)/10
76 LET N$=N$(TO 13)
```

O primeiro problema é que a média calculada pela última parte da linha 80 pode conter muitas casas decimais. Isto é resolvido pela linha 75, por meio de uma operação de *arredondamento* do resultado: multiplicamos a média (M) por 10, tomamos o valor inteiro do resultado e o dividimos por



Utilizado tanto pelo comando *PRINT* como pelo *INPUT*, o ponto e vírgula serve para controlar o modo pelo qual a informação é exibida na tela. Cada item é colocado no vídeo, seguindo-imediatamente o item anterior na mesma linha.

10. Vejamos a média 7,654321: multiplicada por 10, ficaria 76,54321; seu inteiro é 76 que, dividido por 10, é 7,6.

Outro problema é que a primeira nota (NP) será mostrada na tela a partir da coluna 14. Isto quer dizer que o nome do aluno não deve ter mais do que treze caracteres. A função **TO** na linha 77 "amputa" o nome do aluno de modo a caber nesse espaço. Para interromper o programa sem limpar a tela, pressione <**CAPS SHIFT**> junto com <**BREAK**> (no ZX-81, pressione apenas <**BREAK**>). A linha 100 lhe dará tempo de fazer isso.

O programa abaixo oferece o esboço de uma contabilidade doméstica:

```
10 PRINT "NOME";TAB 10;"ITEM" ;
TAB 26;"PRECO";
15 LET total=0
20 FOR I=1 TO 8
35 PRINT AT 20,0;"Data ?"
40 INPUT d$
45 PRINT AT 20,0;"Item ?"
50 INPUT i$
55 PRINT AT 20,0;"Preco?"
60 INPUT p
70 LET total=total+p
80 PRINT d$;TAB 10;i$;TAB 26;p
90 PRINT
100 NEXT I
110 PRINT TAB 26;"------"
120 PRINT TAB 10;"TOTAL";TAB
26; total
```

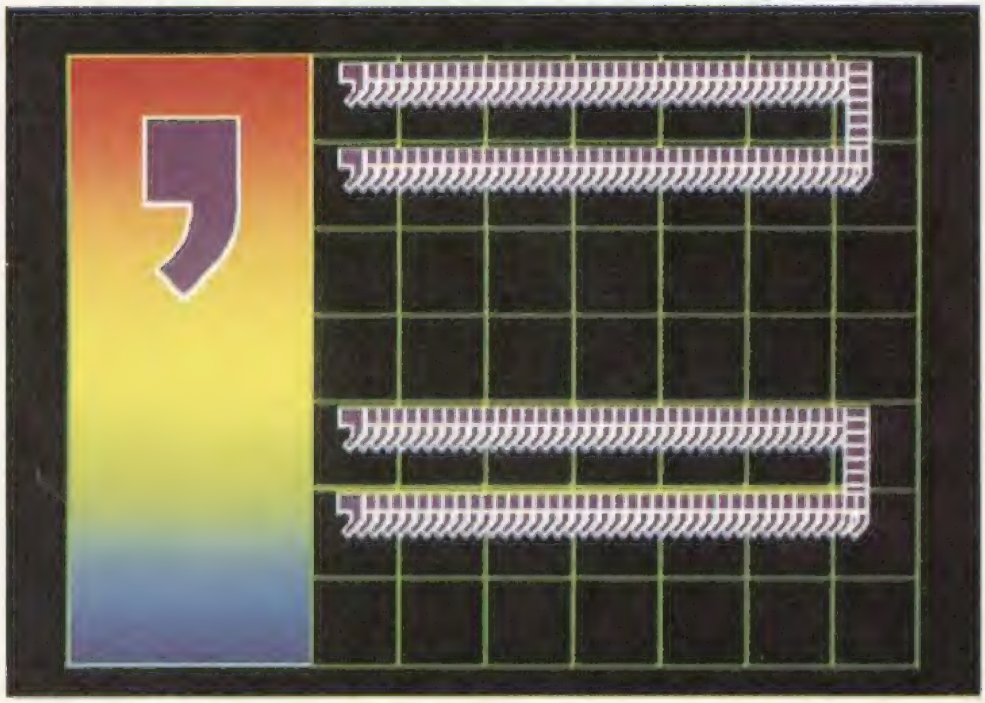
Use corretamente a pontuação, com a declaração **PRINT TAB**. Normalmente, dois pontos e vírgulas são utilizados após **PRINT TAB**:

```
10 FOR n=1 TO 21
20 FOR t=0 TO 24 STEP 6
30 PRINT TAB t;n;
40 NEXT t
50 NEXT n
```

Quando tiver rodado o programa, modifique a linha 30 para:

Veja como usar um ponto e vírgula para preencher a tela com uma mensagem (exemplo rodado no Spectrum).

```
10 PRINT"mensagem"
20 GOTO 10
```



Os apóstrofos instruem o computador a colocar o item seguinte em uma nova linha, quando a máquina estiver impossibilitada de fazer isso automaticamente. Nem todos os computadores, entretanto, contam com esse recurso de pontuação.

```
30 PRINT TAB t,n;
```

... e então compare-a com essa:

```
30 PRINT TAB t;n
```

Um ponto e vírgula significa: "Aproxime bem a próxima palavra, sem nenhum espaço"; uma vírgula significa: "Comece a próxima palavra no início da coluna 0 ou da coluna 15"; uma declaração **PRINT TAB** precisa sempre de dois pontos e vírgulas (do contrário, a desordem se instalará na tela).

TT

A maneira mais fácil de exibir uma mensagem na tela é:

```
PRINT "OLA"
```

Se PRINT "OLA" estiver na última linha da tela, esta rolará para cima, de modo a dar espaço à mensagem (não se esqueça de digitar as aspas). Limpe a tela antes e tudo ficará claro:

```
10 CLS
20 PRINT "OLA"
```

Agora "OLA" aparece no topo à esquerda. Acrescente essa linha e rode o programa:

```
30 PRINT "ADEUS"
```

"ADEUS" aparecerá abaixo de "OLA". Se quiser uma linha branca entre "OLA" e "ADEUS", acrescente:

```
25 PRINT
```

Agora você já sabe como suas mensagens são organizadas verticalmente na tela, mas ainda não sabe como elas aparecerão em cada linha. Suponha que você queira exibir "OLA" no meio da linha. O micro tem uma função (**TAB**) que coloca a mensagem no ponto da linha que você quiser. Acrescente essa linha e rode o programa:

```
40 PRINT TAB(15);"BOA TARDE!"
```

Os espaços em cada linha são numerados de 0 a 31 no TRS-Color, e de 0 a 63 no TRS-80. Então, a mensagem na linha 40 começará no espaço número 15. Não deixe espaço entre **TAB** e o parêntese, no TRS-Color; senão, (15) será tratado como variável e **TAB** não funcionará. Veja como **TAB** calcula o quadrado, o cubo, a raiz quadrada e o inverso dos números de 1 a 12:

```
10 CLS
20 PRINT "NUMERO";TAB(8);"CUBO"
;TAB(14);"QUAD.";TAB(21);"RAIZ"
;TAB(27);"INVER"
30 FOR J=1 TO 12
40 PRINT TAB(1);J;TAB(7);J*J*J;
TAB(13);J*J;TAB(20);INT(SQR(J)*
1000)/1000;TAB(26);INT(1000/J)/
1000
50 NEXT J
```

Um problema que deve ser resolvido neste programa, com um pouco de aritmética elementar, é que a raiz quadrada e o inverso do número J, calculados pela última parte da linha 40, podem ter

muitas casas decimais. Isso é resolvido por meio de uma operação de *arredondamento* do resultado para conter apenas uma casa decimal: multiplicamos a raiz quadrada por 1000, tomamos o valor inteiro do resultado, e o dividimos por 1000. Vejamos, por exemplo, a raiz 7,654321: multiplicada por 1000 ficaria 7654,321; seu inteiro é 7654 que, dividido por 1000, ficaria 7,654. Se você quiser imprimir em um lugar específico na tela deve utilizar **PRINT@** ("PRINT arroba") em vez de **PRINT TAB**.

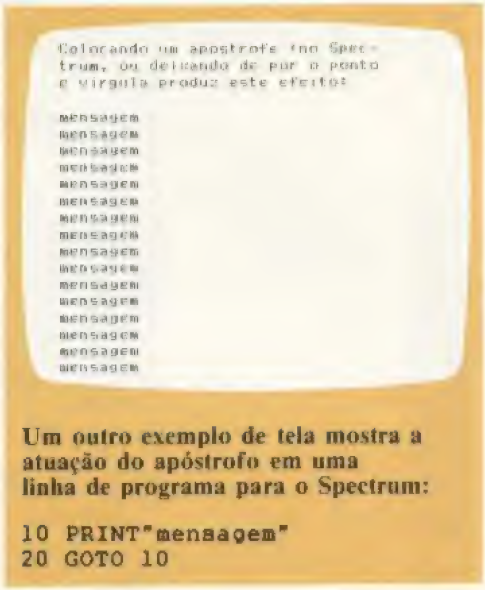
```
10 CLS
20 PRINT @461,"BAIXO"
30 PRINT @77,"ALTO"
40 PRINT @257,"ESQUERDA"
50 PRINT @280,"DIREITA"
```

```
10 CLS
20 PRINT @925,"BAIXO"
30 PRINT @30,"ALTO"
40 PRINT @448,"ESQUERDA"
50 PRINT @502,"DIREITA"
```

Os números após **PRINT@** se referem às locações de tela (numeradas da esquerda para a direita, no TRS). O TRS-80 tem posições numeradas de 0 a 1023, e o TRS-Color (32 colunas) de 0 a 511. Eis um programa que calcula as notas dos alunos de uma classe:

```
 5 CLS
10 PRINT "NOME";TAB(12);"PROVA"
;TAB(18);"ORAL"; TAB(23);
"EXAME";TAB(29);"MED"
20 PRINT @448,"NOME ";
30 INPUT N$
35 PRINT @448,"PROVA";
40 INPUT NP
45 PRINT @448,"ORAL ";
50 INPUT NO
55 PRINT @448,"EXAME";
60 INPUT NE
70 LET M=((NP+NO*2)/3+NE)/2
80 PRINT N$;TAB(14);NP;TAB(19);
NO; TAB(24);NE;TAB(29);M
90 PRINT
100 FOR I=1 TO 400:NEXT I
110 GOTO 20
```

```
 5 CLS
10 PRINT "NOME";TAB(12);"PROVA"
;TAB(18);"ORAL"; TAB(23);
"EXAME";TAB(29);"MED"
20 PRINT @896,"NOME ";
30 INPUT N$
35 PRINT @896,"PROVA";
```



Um outro exemplo de tela mostra a atuação do apóstrofo em uma linha de programa para o Spectrum:

```
10 PRINT"mensagem"
20 GOTO 10
```

```
40 INPUT NP
45 PRINT @896,"ORAL ";
50 INPUT NO
55 PRINT @896,"EXAME";
60 INPUT NE
70 LET M=((NP+NO*2)/3+NE)/2
80 PRINT N$;TAB(14);NP;TAB(19);
NO; TAB(24);NE;TAB(29);M
90 PRINT
100 FOR I=1 TO 400:NEXT I
110 GOTO 20
```

A linha 20 exibe os cabeçalhos das colunas. As linhas 40 a 70 limpam uma linha da tela e perguntam pelos dados necessários (**PRINT@** posiciona a linha que pede a informação na penúltima linha da tela). A linha 80 tabula as informações, e a 90 calcula a média. A variável **PA** faz a máquina checar se a tela foi preenchida.

PONTUAÇÃO

As vírgulas e os pontos e vírgulas controlam a posição do cursor. Agora digite:

```
10 CLS
20 PRINT"O CURSOR RETORNARA"
30 PRINT"SEM O PONTO-E-VIRGULA"
40 PRINT"MAS COM O PONTO-E-
VIRGULA ";
50 PRINT"ACONTECE ISTO"
```

Um ponto e vírgula ao final de uma linha **PRINT** obriga o cursor a permanecer onde estava quando terminou de imprimir. Acrescente essas linhas:

```
60 PRINT"E QUE TAL",
70 PRINT"USAR",
80 PRINT"VIRGULAS",
90 PRINT"?"
```

No TRS-Color: uma vírgula no fim de um **PRINT** faz o cursor saltar para a outra metade da tela. TRS-80: cada vírgula encontrada tabula para a próxima posição).

Os controles de cursor (declaração **LOCATE**), a função **TAB** e o comando **PRINT** com sinais de pontuação servem para melhorar a aparência das exibições na tela do MSX. A maneira mais fácil de mostrar uma mensagem é:

```
PRINT "BOM DIA!"
```

O "BOM DIA" aparecerá à esquerda da tela na próxima linha disponível. Se **PRINT "BOM DIA"** estiver na última linha da tela, esta rolará para cima, de modo a dar espaço à mensagem. Não se esqueça de digitar as aspas, ou obterá uma mensagem de erro. O resultado do comando acima ficará mais claro se você limpar a tela antes:

```
10 CLS
20 PRINT"BOM DIA!"
```

A FUNÇÃO TAB

De todas as instruções de posicionamento do cursor, a função **TAB** é, certamente, a mais utilizada. Ela se comporta, na tela, como o recurso de tabulação das máquinas de escrever, posicionando o cursor alguns espaços a partir da borda esquerda da área de texto da tela. Sua forma pode ser:

```
30 PRINTTAB(15)"BOA TARDE!"
```

Você pode colocar, se quiser, um ponto e vírgula depois do parêntese, mas isso não é necessário. Não deve existir espaço entre **TAB** e seu argumento entre parênteses, cujo valor está no ponto inicial do cordão impresso.

Funções **TAB** múltiplas podem ser utilizadas para posicionar várias observações em cada linha da tela.

```
30 PRINTTAB(10)"BOA" TAB(20)"TA
RDE!"
```

Neste caso, B é impresso na coluna 10, e T, na 20 (o valor do argumento sempre se refere à distância da borda esquerda da área de texto).

TAB é a abreviação de "tabular". Veja no programa a seguir como essa função pode ser útil (ela calcula o quadrado, o cubo, a raiz quadrada e o inverso dos números de 1 a 12):

```
10 CLS
20 PRINT"NUMERO"TAB(10)"QUADRAD
O"TAB(20)"CUBO"TAB(30)"4º POT"
30 FORJ=1TO20
40 PRINTJ;TAB(10)J*J;TAB(20)J*J
*J;TAB(30)J*J*J*J
50 NEXT
```

Ao rodar o programa, você verá uma tabela de números sendo montada. Toda vez que o programa passar pelo laço, uma outra linha de números será exibida na tela (**TAB** permite colocar cada número na coluna correta).

POSICIONE O CURSOR

Suponhamos que você queira imprimir em um lugar específico na tela, e não apenas na próxima linha disponível. Neste caso, **PRINT TAB** não é o mais indicado: use **LOCATE**.

No MSX **LOCATE** pode ser usada para controlar o posicionamento do cursor em qualquer ponto da tela, como ocorre com os comandos **PRINT AT**, **PRINT@** e **HTAB/VTAB** de outras máquinas. Tente o programa abaixo:

```
10 CLS
15 LOCATE 14,23
20 PRINT"SUL"
25 LOCATE 12,1
30 PRINT"NORTE"
35 LOCATE 0,12
40 PRINT"OESTE"
45 LOCATE 25,12
50 PRINT"LESTE"
```

O programa escreve os pontos cardeais junto às quatro margens da tela.

A declaração **LOCATE** define, com dois números, a próxima posição do cursor da tela de textos. O primeiro argumento deve ser um número entre 0 e 39 e diz ao computador da coluna a ser posicionada. O segundo, um número de 0 a 24, indica a posição da linha a ser posicionada. Os argumentos que seguem **LOCATE** podem ser números, variáveis ou expressões com resultado numérico; e somente será considerado o seu valor inteiro (não fracionário) para o cálculo da posição.

Assim, para posicionar uma declaração **PRINT** em um ponto escolhido, estabeleça a posição X (para coluna) e Y (para carreira) e coloque no programa um **LOCATE** antes do **PRINT**.

COMO POSICIONAR ENTRADAS

As entradas de dados através do comando **INPUT** podem ser posicionadas da mesma forma que as mensagens que usam o comando **PRINT**:

```
10 CLS
20 LOCATE 10,10:INPUT"SEU NOME"
;N$
30 LOCATE 12,22:PRINT "OLA, "
:N$;"!"
```

Esse programa imprimirá a primeira instrução na coluna 10 e na linha 10, orientando o usuário com uma mensagem seguida de um ponto de interrogação, para efetuar uma entrada de dados com o **INPUT**. Qualquer coisa que seja digitada será armazenada na variável **N$**. A linha 30 apenas imprime o nome, desta vez na coluna 12 da linha 22. O **LOCATE** não precisa aparecer necessariamente na mesma linha que o **INPUT** e o **PRINT**. Eis aqui um programa que utiliza tanto **LOCATE** quanto **PRINT TAB** e que serve para calcular e exibir as notas dos alunos de uma classe:

```
10 CLS:KEYOFF
20 PRINT"NOME"TAB(10)"NOTA 1"TA
B(20)"NOTA 2"TAB(30)"MEDIA"
25 FORX=1TO15
30 LOCATE0,22:INPUT"NOME ";N$
40 LOCATE0,22:INPUT"NOTA 1";N1
50 LOCATE0,22:INPUT"NOTA 2";N2
60 LOCATE0,X:PRINTN$;TAB(10)N1;
TAB(20)N2;TAB(30)(N1+N2)/2
70 NEXT
```

A linha 20 exibe os cabeçalhos das colunas no topo da tela. As linhas 30 a 50 perguntam pelos dados necessários. **LOCATE** é necessário para posicionar a linha que pede a informação na penúltima linha da tela; sua não inclusão desorganizaria a tabela que está sendo montada na parte de cima. A linha 60 tabula a informação que você acabou de fornecer à máquina e calcula a média.

PONTUAÇÃO

O uso correto dos sinais de pontuação do **PRINT** é importante quando se está exibindo informações na tela. As vírgulas e os pontos e vírgulas controlam a posição do cursor (o quadrado ou o sinal de sublinha que assinala onde aparecem os caracteres).

```
10 CLS
20 PRINT"O CURSOS MUDA DE LINHA
"
30 PRINT"SEM O PONTO-E-VIRGULA"
40 PRINT"MAS, COM ELE ";
50 PRINT"TUDO MUDA!"
```

Quando se inclui um ponto e vírgula ao final de uma linha **PRINT** no programa, o cursor não retorna ao início da próxima linha: ele permanece exatamente onde estava quando terminou de imprimir. A próxima declaração **PRINT** continua diretamente depois da última declaração. Acrescente essas linhas ao programa e rode-o:

```
60 PRINT
70 PRINT"E QUE TAL",
80 PRINT"USAR VIRGULAS?",
90 PRINT"E VER",
100 PRINT"O QUE ACONTECE?"
```

Você provavelmente já conhece este comando:

```
PRINT "BOM DIA!"
```

O seu computador imprimirá a mensagem na primeira linha disponível, começando no limite esquerdo da tela. Se você digitar o programa:

```
10  HOME
20  PRINT "BOM DIA!"
```

verá que a tela será apagada e que a sua mensagem aparecerá na primeira linha, como se tivesse dado uma folha de papel em branco ao computador.

Mas é possível fazer com que a mensagem apareça numa posição diferente. Adicione essa linha:

```
30  PRINT  TAB( 15)"BOA TARDE!"
```

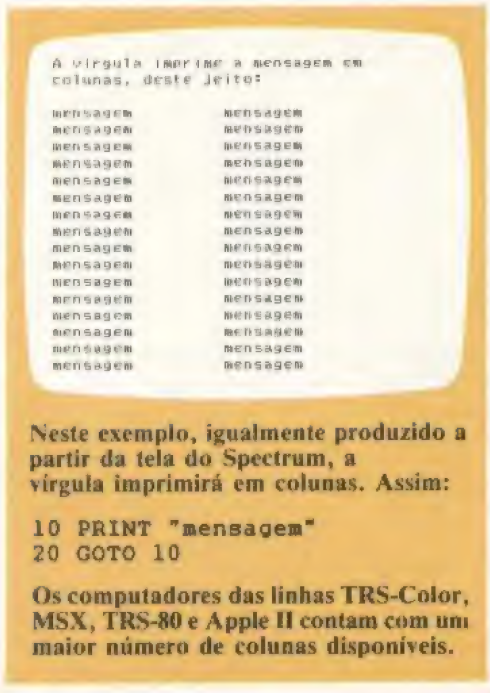
A mensagem aparece centralizada na tela. A primeira letra foi colocada na coluna 15 da linha de impressão. A tela de textos do Apple II tem 24 linhas de quarenta colunas de largura, cada. Você também pode usar mais que um **TAB** por linha:

```
40  PRINT  TAB( 10)"DIA"; TAB(
20)"TARDE"; TAB( 30)"NOITE"
```

E se você quiser pular linhas? É fácil: experimente adicionar as linhas abaixo ao seu programa:

```
25  PRINT
35  PRINT
```

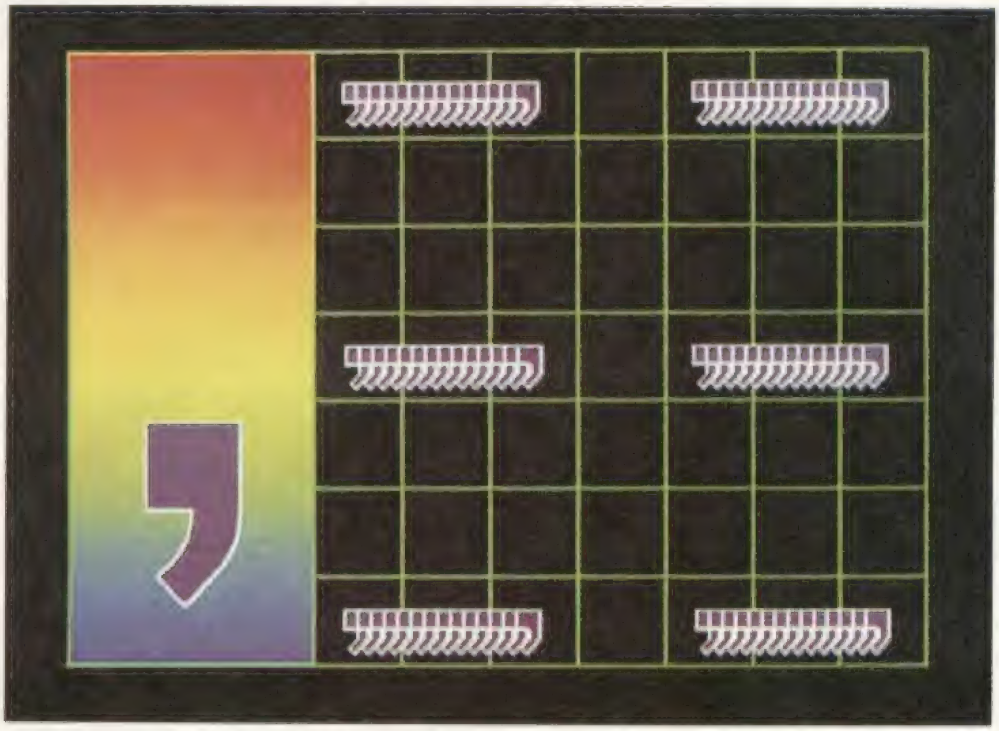
Um comando **PRINT** sem nada depois significa simplesmente uma linha em branco na tela.



Neste exemplo, igualmente produzido a partir da tela do Spectrum, a vírgula imprimirá em colunas. Assim:

```
10 PRINT "mensagem"
20 GOTO 10
```

Os computadores das linhas TRS-Color, MSX, TRS-80 e Apple II contam com um maior número de colunas disponíveis.



Nas declarações *PRINT* e *INPUT*, as vírgulas indicam que a informação será exibida em campos. O ZX-81, o Spectrum e o TRS-Color têm dois campos de dezesseis colunas; o MSX e o Apple II, três campos de oito colunas; já o TRS-80 conta com quatro campos.

HTAB E VTAB

O Apple II tem outro sistema para posicionar mensagens na tela.

```
10 HOME
30 HTAB 15 : PRINT "BOA TARDE"
```

Observe que o efeito é semelhante ao do **PRINT TAB**. Acrescente a linha a seguir e rode o programa novamente:

```
20 VTAB 15
```

A sua mensagem está agora no centro da tela. Com esses dois comandos, **HTAB** (abreviatura de *horizontal tabulation*) e **VTAB** (*vertical tabulation*), você pode colocar o cursor (e suas mensagens) em qualquer lugar da tela.

TABULE AS ENTRADAS

As entradas de dados através do **INPUT** podem ser posicionadas da mesma forma que as mensagens com **PRINT**:

```
10  HOME
20  VTAB 10: HTAB 10: INPUT "SE
U NOME? ";N$
30  VTAB 22: HTAB 12: PRINT "OL
A, ";N$;"!"
```

A primeira instrução será impressa na coluna 10 e na linha 10, orientando o usuário com uma mensagem, seguida de um ponto de interrogação, para entrar dados com o **INPUT**. Qualquer coisa que seja digitada será armazenada na variável **N$**. A linha 30 imprime o nome, desta vez na coluna 12 da linha 22. A ordem em que **VTAB** e **HTAB** aparecem no programa não é importante. Se você não quiser um ponto de interrogação, omita-o da linha 20.

```
INPUT"ENTRE O CUSTO EM CZ$:";
CUSTO
```

Um ponto de interrogação prejudicaria a clareza da tela. O programa abaixo serve para calcular e exibir as notas dos alunos de uma classe:

```
10  HOME
20  INPUT "QUANTOS ALUNOS? ";A
30  HOME : PRINT "NOTA1","NOTA2
","MEDIA"
40  FOR J = 1 TO A
50  VTAB 21: INPUT "NOTA1? ";N1

60  VTAB 21: INPUT "NOTA2? ";N2

70  VTAB J + 3: PRINT N1,N2,(N1
 + N2) / 2
80  NEXT
```

A linha 20 exibe os cabeçalhos das colunas no topo da tela. As linhas 40 a 70 limpam uma linha e perguntam pelos dados necessários. O **HTAB** e o **VTAB** antes do **INPUT** são necessários para posicionar a linha que pede a informação na penúltima linha da tela.

A linha 80 tabula a informação que você acabou de fornecer à máquina, e a linha 90 calcula a média.

A média das notas pode dar um resultado com muitas casas decimais (uma dízima periódica, por exemplo). Ora, esse resultado não caberia na linha de tela do Apple II, extravasando para a linha seguinte e "estragando" o aspecto da tela. Isso pode ser resolvido por meio de uma operação de arredondamento para que o resultado contenha apenas uma casa decimal: multiplicamos a raiz quadrada por 1000, tomamos o valor inteiro do resultado, e o dividimos por 1000. Vejamos, por exemplo, a média 7,654321: multiplicada por 1000 ficaria 7654,321; seu inteiro é 7654 que dividido por 1000 ficaria 7,654. Se você quiser um resultado com duas casas decimais, ao invés de três, multiplique e divida por 100. Substitua a linha:

```
70 VTAB J + 3: PRINT N1,N2,
INT((N1+N2)/2/10)*10
```

PONTUAÇÃO

Em BASIC, a pontuação oferece três possibilidades: a vírgula, o ponto e vírgula e a ausência de pontuação. O ponto e vírgula faz com que o cursor não mude de posição após imprimir uma mensagem na tela, o que resulta em uma justaposição, se várias delas forem impressas em seqüência:

```
10  HOME
20 A = 10:B = 20:C = 30
30  PRINT A;B;C
```

A vírgula produz um efeito bem diferente. Tente adicionar essa linha e veja o que acontece:

```
40  PRINT A,B,C
```

A tela, neste caso, é dividida em três colunas de tabulação automática e tudo que for mostrado na tela obedecerá a essa tabulação, exceto se algum dado exceder o tamanho da coluna.

Neste programa mostramos como a tabulação automática com a vírgula pode ser útil (ele calcula o quadrado e o cubo dos números de 1 a 20):

```
10  HOME
20  PRINT "NUMERO","QUADR","CUB
O"
25  PRINT
30  FOR J = 1 TO 20
40  PRINT J,J * J,J * J * J
50  NEXT
```

Você verá uma tabela de números sendo montada. Toda vez que o programa passar pelo laço uma outra linha de números será exibida na tela. A utilização do **TAB** permitirá que você coloque cada número na coluna correta.

TORNE O JOGO MAIS DIFÍCIL

- COMO DESENHAR UM LABIRINTO ALEATÓRIO
- DUAS MANEIRAS DE TORNAR O JOGO MAIS DIFÍCIL
- DESCUBRA O TESOURO

Se você gosta de enfrentar desafios, aprenda a tornar mais complicados seus jogos de labirinto, incorporando a eles níveis crescentes de dificuldade. E veja quanto tempo leva para encontrar um tesouro oculto no esconderijo de um sangüinário pirata.

Muitos programas pedem que se escolha um nível de dificuldade, antes de começar o jogo. Essa exigência permite que tanto os iniciantes quanto os especialistas possam utilizar o mesmo jogo, sem que este se torne excessivamente fácil ou difícil. Existem muitas maneiras de se introduzir níveis de dificuldade em um jogo, dependendo da sua natureza. Por exemplo: podemos mudar o número de inimigos enfrentados pelo jogador; desenvolver a ação em diferentes velocidades; conceder maior ou menor tempo para o jogo; variar os problemas a serem encontrados pelo jogador, e assim por diante.

Desta vez, vamos aprender a incorporar níveis de dificuldade a um programa de jogo de labirinto. O jogo escolhido emprega uma entre duas maneiras possíveis de se produzir níveis de dificuldade — o método a ser utilizado em detalhe dependerá do modelo de seu computador. O jogo não envolve apenas a tentativa de se achar o caminho através do labirinto, mas também uma limitação no tempo de que dispõe o jogador para guiar um homenzinho até um tesouro desenhado em algum lugar do labirinto. Não apresentaremos aqui uma versão do programa para microcomputadores da linha ZX-81, pois o jogo ficaria excessivamente lento. Também não será apresentada versão para micros da linha TRS-80, devido à baixa resolução gráfica do vídeo desses modelos.

Existem basicamente duas formas de aumentar a dificuldade de resolução de um labirinto: ou se altera o limite de

tempo, ou se varia a complexidade do labirinto. A razão pela qual devemos escolher métodos diversos, de acordo com o tipo do computador, tem a ver com a maneira como o labirinto é produzido. Esse exemplo mostra que você precisa decidir o caminho a seguir, quando quiser inventar jogos com diferentes níveis de dificuldade.

AS VIDAS

Uma das maneiras de tornar mais excitante um jogo de labirinto consiste em impor algum tipo de penalidade ao jogador que esgota seu tempo sem encontrar o tesouro. Pode-se, por exemplo, fazer com que ele perca alguns pontos na contagem; mas a penalidade mais comum é obrigá-lo a perder uma "vida".

Nesse jogo daremos três "vidas" ao jogador: se ele não achar o tesouro dentro do limite de tempo, em três tentativas, o jogo terminará.

LABIRINTOS ALEATÓRIOS

O jogo é baseado em uma sub-rotina de produção de labirintos aleatórios (ou seja, de desenho gerado ao acaso). É um programa muito interessante, pois desenha um labirinto totalmente diferente de cada vez, evitando assim que você tenha de projetar uma série enorme desses desenhos. Já mostramos, em um artigo anterior, como projetar labirintos, utilizando linhas **DATA** para incorporar o seu desenho ao programa. Imagine, então, como seria complicado incluir vários labirintos ao programa, usando essa técnica.

A geração aleatória de labirintos é muito mais fácil do quê isto, embora apresente algumas dificuldades complementares. Uma maneira óbvia de projetá-los, por exemplo, seria desenhar blocos gráficos aleatoriamente na tela. O problema, neste caso, é que nada garante que se formem caminhos interiores entre os blocos; ou seja, pode não surgir um verdadeiro labirinto. Assim, caso escolhesse esse método, você teria que encontrar maneiras de checar a existência de, no mínimo, uma saída.

LABIRINTOS E CAMINHOS AO ACASO

O melhor modo de se desenhar labirintos aleatórios é inventar um programa que trace caminhos ao acaso, incluindo-os depois no desenho final. O programa proposto a seguir foi desenvolvido de tal forma que a linha do caminho aleatório fica contida no interior da moldura desenhada na tela. Não é permitido que a linha cruze com ela mesma. Quando o caminho aleatório não puder prosseguir, o programa fará o mesmo percurso de volta. Ele faz isso passo a passo, examinando a área em torno do caminho anteriormente traçado, até encontrar espaço livre para prosseguir. Quando esse espaço é encontrado, o programa inicia um outro desvio no caminho aleatório, prosseguindo por ele até ser imobilizado novamente, e assim por diante. O computador continua tentando desenhar novos caminhos até que toda a tela seja preenchida; nesse momento, ele volta ao ponto de partida.

Quando o programa acaba de desenhar o labirinto, apenas um caminho livre aparece na tela; esse caminho pode ser facilmente distinguido, pois os desvios não são muito complicados. O labirinto também será solucionável através da *regra da mão direita*. De acordo com essa regra, é sempre possível sair de um labirinto, seguindo-se a sua parede direita (ou esquerda, tanto faz, desde que seja sempre a mesma). Para impedir que alguém aplique a regra, basta construir algumas *ilhas* no labirinto, obrigando o pobre jogador a ficar dando voltas sem fim. Assim, após desenhar um labirinto, o programa traça um certo número de blocos aleatórios que fazem com que o desenho pareça mais complexo, impedindo que alguém recorra à regra da mão direita.

Uma vez digitado, guarde o programa em fita ou disco, pois o próximo artigo mostrará como acrescentar a ele alguns efeitos sonoros.

SE

O programa para os micros da linha Spectrum começa estabelecendo os blocos gráficos, introduzindo as variáveis e fazendo uma preparação geral para o jogo. Digite essa seção do programa, mas não a execute ainda:

```
10 FOR n=0 TO 23: READ a:
POKE USR "a"+n,a: NEXT n
20 LET hs=0
30 INPUT "Selecione nivel (1
a 6) ";ta
40 LET ta=1100-100*ta
50 BORDER 1: PAPER 1: INK 0:
CLS : INK 7
60 LET s=0: LET vidas=3
70 PRINT  BRIGHT 1; PAPER 6;
INK 2;" SCORE        RECORDE
      "
490 DATA 24,24,60,82,82,24,36,
36,127,65,93,85,81,95,64,127,
24,24,255,255,24,24,24,24
```

A linha 10 define os blocos gráficos para o jogo — um homenzinho, o tesouro e uma cruz — por meio da leitura dos dados na linha 490. A variável que armazenará o recorde — **HS** (*high score*) — é zerada na linha 20.

A seguir, pede-se ao jogador para escolher um nível de dificuldade de 1 a 6 (**TA**). Quanto menor for o número, mais baixo será esse nível (e, portanto, mais fácil será o jogo). Você verá que na linha 40 os números mais baixos estabelecem tempos maiores, e os números mais altos, tempo menores.

As cores da tela e dos gráficos são estabelecidas na linha 50. A linha 60 zera o placar e atribui as três "vidas" ao jogador. A linha 70, finalmente, exibe as palavras ESCORE (contagem) e RECORDE, juntamente com os espaços em branco para colocar os números correspondentes, na tela.

DESENHE O LABIRINTO

Agora, digite essas linhas:

```
 80 FOR n=22561 TO 22589: POKE
 n,16:   POKE n+640,16: NEXT n
 90 FOR n=1 TO 21: POKE 22528+
 n*32,16  : POKE 22558+n*32,16
 : POKE 22559+n*32,9: NEXT n
100 LET b=22593: LET a=b
110 DIM a(4): LET a(1)=-1: LET
a(2)=- 32: LET a(3)=1: LET a(4
)=32
120 POKE a,56
130 LET j=INT (RND*4)+1: LET g
=j
140 LET b=a+a(j)*2: IF PEEK b=
8 THEN  POKE b,j: POKE a+a(j),
56: LET a=b : GOTO 130
150 LET j=j+1: IF j=5 THEN
LET j=1
160 IF j<>g THEN  GOTO 140
170 LET j=PEEK a: POKE a,56:
IF j<5   THEN  LET a=a-a(j)*2:
GOTO 130
180 POKE 22625,56
190 FOR n=1 TO 20
200 LET k=22528+64*(INT (RND*9
)+2)+   INT (RND*29)+1
210 POKE k,56: NEXT n
```

A borda do labirinto é traçada pelas linhas 80 e 90, colocando-se o código 16 por meio de comandos **POKE** nas áreas de memória de vídeo. Esse código define a cor de fundo (**PAPER**), gerando portanto uma borda constituída de blocos vermelhos. As linhas 100 a 180 desenham o labirinto. Não caia na tentação de interromper o programa e limpar a tela nesse ponto. Se você fizer isso, o labirinto se perderá, pois ainda está armazenado apenas no arquivo de atributos. Para completar o labirinto, as linhas 190 a 210 exibem vinte quadrados em posições aleatórias dentro do labi-

rinto. Se um quadrado "cair" em cima de uma parede, uma passagem será automaticamente aberta.

COMO CRIAR UM JOGO

A próxima seção do programa diz respeito ao jogo em si (não tente ainda rodar o programa):

```
220 LET x=15: LET y=10
230 LET tx=INT (RND*15)*2+1
240 LET ty=INT (RND*10)*2+2
250 PRINT  BRIGHT 1; PAPER 2;
AT 0,7;s;AT 0,24;hs
260 POKE 23672,0: POKE 23673,0
270 PRINT  FLASH 1; PAPER 3;
INK 6;AT ty,tx;CHR$ 145
280 PRINT  INK 2; PAPER 7;AT y
,x;CHR$ 144
290 IF PEEK 23672+256*PEEK
23673>ta THEN  GOTO 390
300 IF INKEY$="" THEN  GOTO
290
310 LET a$=INKEY$: LET sx=x:
LET sy=y
320 IF a$="z" AND ATTR (y,x-1)
>=56 THEN  LET x=x-1
330 IF a$="x" AND ATTR (y,x+1)
>=56 THEN  LET x=x+1
340 IF a$="k" AND ATTR (y-1,x)
>=56 THEN  LET y=y-1
350 IF a$="m" AND ATTR (y+1,x)
>=56 THEN  LET y=y+1
360 PRINT  PAPER 7; INK 2;AT s
y,sx;" ";AT y,x;CHR$ 144
370 IF ty=y AND tx=x THEN
GOTO 470
380 GOTO 290
```

A posição inicial do homenzinho é estabelecida pela linha 220; o homenzinho começa sempre nas posições 15,10 no início de cada jogo.

O tesouro é colocado aleatoriamente pelas linhas 230 e 240, e a linha 270 o exibe na tela. A gama de escolha dos números aleatórios garante que o tesouro caia sempre em um caminho.

A linha 250 exibe os valores do escore e do recorde (ambos inicialmente em 0), ao passo que a linha 260 zera o cronômetro interno por meio de dois **POKE** em duas locações de memória. A linha 290 verifica se o limite de tempo foi excedido e, se isso aconteceu, pula para a linha 390.

O homenzinho é exibido pela linha 280. Note que os comandos **CHR$ 144** e **CHR$ 145** das linhas 270 e 280 são utilizados para exibir os blocos gráficos predefinidos para esses códigos. O **CHR$** é empregado aqui porque é mais fácil de ser visto na listagem do programa, embora você possa utilizar o método alternativo através de letras, como foi explicado anteriormente. As linhas restantes (ou seja, da 300 à 380) se ocupam com a movimentação do homenzinho no labirinto. As linhas 320 a 350 verificam se está sendo pressionada a tecla adequada, e se o próximo quadrado para a movimentação é um caminho e não uma parede. O teste utiliza o **ATTR**, que já foi explicado anteriormente. A linha 260 anula a última posição do homenzinho e o exibe novamente em uma posição diferente. A linha 370 testa se o homenzinho atingiu o tesouro. Se ele atingiu, a contagem de pontos é aumentada, antes de se sortear e exibir um outro tesouro a ser encontrado.

A MORTE DO PIRATA

A última seção final do programa é apresentada a seguir. Agora, finalmente, você pode rodá-lo.

```
390 PRINT  FLASH 1; PAPER 0;
INK 5;AT y,x;CHR$ 146
400 LET vidas=vidas-1: FOR f=1
TO 200: NEXT f: IF vidas>0
THEN  GOTO 260
410 IF s>hs THEN  LET hs=s
```

```
420 PRINT  BRIGHT 1; PAPER 2;
AT 0,24;hs
430 PRINT  FLASH 1;AT 10,1;" P
ressione qualquer tecla para
jogar de novo "
440 IF INKEY$<>"" THEN  GOTO
440
450 IF INKEY$="" THEN  GOTO
450
460 GOTO 30
470 LET s=s+ta-PEEK 23672+256*
PEEK 23673: GOTO 230
```

Quando o jogador perde uma "vida", e antes mesmo que a linha 400 subtraia 1 ao total de "vidas", uma cruz piscante (**CHR$ 146**) é exibida pela linha 390. Existe ainda uma pausa antes que o jogo retorne à linha 260, a qual coloca novamente o relógio em 0, de modo a ficar pronto para outra tentativa de atingir o tesouro. Esse retorno, evidentemente, só ocorrerá se o jogador ainda tiver mais "vidas" para usar.

Se não restar nenhuma "vida", a linha 410 compara o placar final com o último recorde. O recorde registrado é alterado quando o escore for maior. Em ambos os casos, a linha 420 exibe o recorde obtido até o momento.

A linha 410 é necessária para bloquear o jogo, caso o jogador ainda esteja pressionando uma das teclas de movimentação. A linha 420, por sua vez, bloqueia o programa até que uma nova pressão em qualquer tecla assinale que o jogador quer prosseguir.

A linha 470 calcula o escore, logo após o encontro do homem com o tesouro. O teste está na linha 370.

T

A tela de texto proporciona a maneira mais fácil de se desenhar um labirinto nos micros da linha TRS-Color. Mas os labirintos assim obtidos seriam muito simplificados por causa do grande tamanho e do pequeno número de blocos gráficos disponíveis.

Em vez disso, o programa aqui proposto desenha os labirintos na tela com gráficos de alta resolução. Embora seja mais complicado do que o que se pode conseguir na tela de textos, ele tem a vantagem de desenhar labirintos de diferentes complexidades, fornecendo assim vários níveis de dificuldade.

Imagine que o caminho seja formado por uma série de blocos quadrados. Se você quisesse desenhar um labirinto bem simples, bastaria escolher um bloco grande, ao passo que, se quisesse um labirinto mais complexo, deveria optar por um bloco de menor tamanho.

A primeira seção do programa introduz as variáveis e prepara o computador para desenhar labirintos aleatórios. Digite o programa, mas não execute ainda, senão você obterá um erro do tipo UL — linha indefinida —, quando o programa tentar executar o **GOSUB 1000** da linha 110.

```
10 PMODE 4,1
20 CLS:PRINT @193,"NIVEL DE DIF
ICULDADE (0-5)";
30 L$=INKEY$:IF L$<"0" OR L$>"5
" THEN 30
40 BS=12-VAL(L$):NX=2*INT(.5+12
8/BS):NY=2*INT(.5+96/BS)
50 SX=250-BS*NX:SY=190-BS*NY
60 DIMP(NX,NY),A(5),B(5)
70 PCLS 5: DRAW"S"+STR$(INT(8.5
-4*VAL(L$)/5))+"C0BM0,0BR2BDNFN
GD3NFG"
80 GET (0,0)-(BS-1,BS-1),A,G
90 GET(10,10)-(BS+9,BS+9),B,G:C
OLOR 5,0
100 CLS:PRINT @226,"GERANDO LAB
IRINTO DE NIVEL ";L$
110 GOSUB 1000
120 GOTO 120
```

A linha 10 diz ao computador para utilizar o quarto modo gráfico (**PMODE 4**) para todo o programa. A tela de alta resolução não é ligada nesse estágio. A linha 20 exibe, então, a mensagem NÍVEL DE DIFICULDADE (0 - 5).

L$** é o nível que o jogador escolheu. O valor numérico de **L$ — VAL (L$) na linha 40 — regula o tamanho do bloco, a extensão do caminho e a complexidade do labirinto. O **INKEY$** na linha 30 significa que o jogador não precisa digitar mais que o único dígito em resposta à pergunta, e que o programa continua sem que ele precise pressionar **<ENTER>**.

Na linha 40, o **BS** é o tamanho do bloco gráfico, em pixels. Esse tamanho pode variar de sete a doze pixels. **NX** é o número de blocos na direção horizontal, e **NY** é o número de blocos na direção vertical.

Antes de desenhar o labirinto na tela, o computador calculará sua aparência final e colocará essa informação no conjunto P, o qual é dimensionado (**DIM**) na linha 60. O conjunto A contém o formato do homenzinho, e o conjunto B, um espaço em branco, de modo a provocar sua animação gráfica. O homenzinho é desenhado pela linha 70. Nesse programa o homenzinho será desenhado em tamanho maior, quando o caminho do labirinto for mais largo, e menor quando for mais estreito.

Agora que o homenzinho foi desenhado, a linha 80 o coloca no conjunto A por intermédio de um comando **GET**, e a linha 90 preenche o conjunto B de branco. Nas lições anteriores, quando um espaço em branco era utilizado em um jogo, não era necessário colocar nada no conjunto: tínhamos apenas um conjunto vazio. Desta vez, é necessário um espaço em branco, de modo a com-

binar com a cor de fundo do caminho.

O comando **COLOR** na linha 90 assegura que o labirinto será desenhado mais tarde na cor correta (de outro modo você estaria desenhando um labirinto preto sobre um fundo preto).

A linha 100 diz ao jogador que o labirinto está sendo produzido; pode demorar algum tempo até o desenho aparecer.

DESENHE O LABIRINTO

Agora digite a sub-rotina de produção do labirinto — chamada pela linha 110 — e você poderá rodar o programa:

```
1000 FOR J=0 TO NX:P(J,NY)=6:P(
J,0)=6:NEXT
1010 FOR J=0 TO NY-2:P(0,J)=6:P
(NX,J)=6:NEXT
1020 X=2:Y=2:LX=2:LY=2
1030 J=RND(4)-1:G=J
1040 Y=LY+2*((J=0)-(J=2)):X=LX+
2*((J=3)-(J=1))
1050 IF P(X,Y)=0 THEN P(X,Y)=J+
1:P((X+LX)/2,(Y+LY)/2)=5:LX=X:L
Y=Y:GOTO 1030
1060 J=(J+1)AND 3: IF J<>G THEN
 1040
1070 J=P(LX,LY)-1:P(LX,LY)=5:IF
 J<4 THEN LX=LX-2*((J=3)-(J=1))
:LY=LY-2*((J=0)-(J=2)):GOTO 103
0
1080 FOR J=0 TO 20:P(2+2*RND((N
X-3)/2),1+RND(NY-3))=5:P(1+RND(
NX-3),2+2*RND((NY-3)/2))=5:NEXT
1090 SCREEN 1,1:PCLS
1100 FOR J=2 TO NX-2:FOR K=2 TO
 NY-2
1110 IF P(J,K)=5 THEN LINE(J*BS
+SX,K*BS+SY)-((J+1)*BS+SX-1,(K+
1)*BS+SY-1),PSET,BF
1120 NEXT K,J:RETURN
```

As linhas 1000 a 1080 "desenham" o labirinto na memória do computador e armazenam a sua forma no conjunto P — cada elemento do conjunto corresponde a um bloco do labirinto. A sub-rotina armazena o número 5 em P onde quer que exista um caminho, e o número 0, se existir uma parede. Uma vez que o labirinto tenha sido armazenado em P, a linha 1090 liga a tela de alta resolução e a deixa pronta para a execução do desenho.

As linhas 1100 e 1120 exibem o labirinto na tela mediante a verificação dos conteúdos de P. Quando um 5 for encontrado, um quadrado branco será impresso.

MOVIMENTO, TESOURO, VIDAS

Agora você precisa de um jogo para acompanhar o seu labirinto aleatório. Digite a próxima seção do programa, mas não o rode porque ele chamará uma sub-rotina que ainda não existe.

```
120 X=2:Y=2:LX=2:LY=2:TI=800:LI
=3
130 TIMER=0
140 X1=1+RND(NX-3):Y1=1+RND(NY-
3):IF P(X1,Y1)=5 THEN P(X1,Y1)=
7:DRAW "S4C0BM"+STR$(SX+X1*BS)+
","+STR$(SY+Y1*BS)+"BFR5D5L3U3H
D" ELSE 140
150 X1=X*BS+SX:Y1=Y*BS+SY
160 PUT(X1,Y1)-(X1+BS-1,Y1+BS-1
),A,PSET
170 IF PEEK (338)=251 THEN Y=Y-
1
180 IF PEEK (342)=253 THEN Y=Y+
1
190 IF PEEK (340)=247 THEN X=X-
1
200 IF PEEK (338)=247 THEN X=X+
1
210 IF P(X,Y)=7 THEN F=1:P(X,Y)
=5: GOTO 230
220 IF P(X,Y)<>5 THEN X=LX:Y=LY
:GOTO 170
230 IF X<>LX OR Y<>LY THEN PUT
(X1,Y1)-(X1+BS-1,Y1+BS-1),B,PSE
T:LX=X:LY=Y:FOR P=1 TO 68:NEXT
240 IF F=1 THEN F=0:SC=SC+(TI-T
IMER):TI=TI-10:GOTO 130
250 IF TIMER >TI THEN GOSUB 500
: IF LI<1 THEN 100
260 GOTO 150
```

A linha 120, que substitui a linha 120 da seção anterior, contém as variáveis a partir das quais é calculada a posição do homenzinho. X, Y é a localização atual do homenzinho, e LX e LY, sua última posição; porém, em virtude da largura variável do caminho, os valores devem ser ligeiramente ajustados um pouco antes de o homenzinho aparecer na tela. **TI** é o limite de tempo para descobrir o tesouro. Esse tempo limite é de dezesseis segundos. Se o homenzinho não tiver encontrado o tesouro, quando o tempo se esgotar, perderá uma "vida". No início do jogo, o jogador tem três "vidas" — **LI=3**.

O cronômetro interno do computador é zerado na linha 130, antes que a linha 140 selecione uma posição aleatória para o tesouro. O elemento correspondente em P é examinado para que haja certeza de que nesse lugar há um caminho e não uma parede. Se o tesouro estiver em um caminho, o valor do conjunto será modificado de 5 para 7. A última parte da linha desenha o tesouro no labirinto.

A posição do homenzinho é calculada na linha 150, levando-se em consideração a largura do caminho, que é o tamanho do bloco BS. A linha 160 coloca o homenzinho em posição.

As linhas 170 a 220 examinam o teclado por meio de comandos **PEEK**, de modo a permitir a movimentação do pirata no labirinto. Como não é possível deixá-lo atravessar paredes, a linha 220 o mantém dentro do caminho. A linha 210 verifica se o tesouro foi encontrado, examinando o elemento correspondente em P para o número 7. Se o computador o achar, o "indicador de descoberta" (F) será igualado a 1.

A linha 230 movimenta o pirata. O espaço em branco é colocado, com o comando **PUT**, sobre a última posição do personagem, e sua localização atual torna-se a última posição.

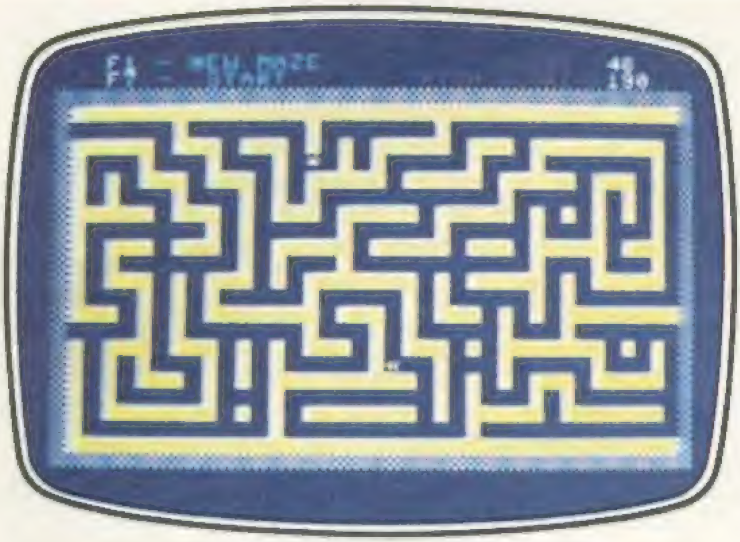
A linha 240 calcula o escore, caso o tesouro tenha sido encontrado. O limite de tempo é diminuído então de 10 segundos. O F volta ao 0, e o programa retorna para zerar o cronômetro. O programa continua, redesenhando o tesouro em outro lugar, deixando o pirata no mesmo ponto em que estava quando o último tesouro foi encontrado.

Caso o tesouro demore muito a ser encontrado, a linha 250 chamará a sub-rotina que se inicia na linha 500. Se, mesmo não tendo achado o tesouro, o homenzinho ainda dispuser de algum tempo, a linha 260 retornará ao programa, calculando sua nova posição.

A EXIBIÇÃO DO ESCORE

Esta é a sub-rotina final. Ela exibirá o escore e o número de "vidas", depois que uma "vida" for perdida:

```
500 CLS:SCREEN 0,0:LI=LI-1
510 PRINT @106,"NIVEL=";LS
520 IF LI>0 THEN PRINT @202,"VI
DAS=";LI
530 PRINT @298,"SCORE=";SC
540 IF LI>0 THEN FOR J=1 TO 600
0:NEXT:TIMER=0:SCREEN 1,1:RETUR
N
550 PRINT @358,"QUER OUTRA VEZ
(S/N)?"
```



Um pirata à procura de um fabuloso tesouro no labirinto do MSX.

Outro bucaneiro explora o colorido labirinto do Spectrum.

```
560 A$=INKEY$:IF A$<>"S" AND A$
<>"N" THEN 560
570 IF A$="S" THEN RUN
580 END
```

Agora você pode rodar o programa. Se uma "vida" for perdida, o programa religa a tela de texto — **SCREEN 0,0.** A linha 500 também diminui de 1 o número de "vidas" restantes do jogador.

As linhas 510 a 530 exibem o nível de dificuldade, o número de "vidas" restantes (se o jogo continuar) e a contagem de pontos. Se ainda sobrarem algumas "vidas", a linha 540 introduzirá uma pausa antes de zerar o cronômetro, religando a tela de alta resolução e retornando à sub-rotina.

As linhas 550 e 580 perguntam se o jogador quer tentar de novo, e ele ou pára o programa ou roda novamente. O **RUN** é utilizado na linha 570 para limpar P para um novo labirinto.

MSX

A versão do programa para os micros da linha MSX desenha os labirintos na tela com gráficos de alta resolução. Embora mais complexa do que o programa para se desenhar na tela de textos, ela tem a vantagem de poder traçar labirintos de diferentes complexidades, fornecendo assim uma variação no nível de dificuldade.

O caminho aleatório é formado por uma série de blocos quadrados. Se você quisesse desenhar um labirinto bem simples, bastaria escolher um bloco grande, ao passo que, se quisesse um labirinto mais complexo, teria que optar por um bloco de menor tamanho.

A primeira seção do programa introduz as variáveis e prepara o computador, de um modo geral, para desenhar labirintos aleatórios. Digite o programa, mas não o execute ainda, pois não aparecerá nada na tela, por enquanto.

```
10 SCREEN 0
20 LOCATE 2,11:PRINT "Nível de
dificuldade (0-5) ?"
30 L$=INKEY$:IF L$<"0" OR L$>"5
" THEN 30
40 BS=13-VAL(L$):NX=2*INT(.5+12
8/BS):NY=2*INT(.5+96/BS)
50 SX=250-BS*NX:SY=190-BS*NY
60 DIM P(NX,NY)
70 FOR I=1 TO 8
80 READ A,B,C:A$=A$+CHR$(A)
84 B$=B$+CHR$(B)
88 C$=C$+CHR$(C)
90 NEXT
100 CLS:LOCATE 2,11
105 PRINT "Gerando labirinto de
 nível ";L$
110 DATA 24,24,127,24,24,65,60,
255,93,82,255,85,82,24,81,24,24
,95,36,24,64,36,24,127
```

A linha 10 diz ao computador para utilizar a tela de textos (**SCREEN 0**). A linha 20 exibe, então, a mensagem: "Nível de dificuldade (0 - 5) ?"

L$** é o nível que o jogador escolheu. O valor numérico de **L$ — VAL (L$) na linha 40 — regula o tamanho do bloco, a extensão do caminho e a complexidade do labirinto. O INKEY$ na linha 30 significa que o jogador não precisa digitar mais que um único dígito em resposta à pergunta, e que o programa continua sem que ele precise pressionar <ENTER>.

Na linha 40, o BS é o tamanho do bloco gráfico, em pixels. O tamanho pode variar de oito a doze pixels. NX é o número de blocos na direção horizontal, e NY é o número de blocos na direção vertical.

Antes de desenhar o labirinto na tela, o computador calculará sua aparência final e colocará essa informação no conjunto P, que é dimensionado (**DIM**) na linha 60. A variável **A$** contém o formato do homenzinho, e a variável **B$**, uma cruz, para assinalar sua "morte". A variável **C$** contém a imagem do tesouro. Os códigos gráficos correspondentes são lidos em **DATA**, na linha 110, e concatenados nas variáveis alfanuméricas mencionadas. Posteriormente, esses blocos gráficos serão colocados dentro de sprites.

O computador limpa a tela de textos na linha 100, e esta informa que o labirinto está sendo produzido.

DESENHE O LABIRINTO

Agora, digite a parte do programa que produz o labirinto, e rode-o para ver o resultado:

```
1000 FOR J=0 TO NX
1005 P(J,NY)=6:P(J,0)=6
1008 NEXT
1010 FOR J=0 TO NY-2
1014 P(0,J)=6:P(NX,J)=6
1018 NEXT
1020 X=2:Y=2:LX=2:LY=2
1030 J=INT(RND(1)*4):G=J
1040 Y=LY+2*((J=0)-(J=2)):X=LX+
2*((J=3)-(J=1))
1050 IF P(X,Y)=0 THEN P(X,Y)=J+
1:P((X+LX)/2,(Y+LY)/2)=5:LX=X:L
Y=Y:GOTO 1030
1060 J=(J+1) AND 3:IF J<>G THEN
 1040
1070 J=P(LX,LY)-1:P(LX,LY)=5
1075 IF J<4 THEN LX=LX-2*((J=3)
-(J=1)):LY=LY-2*((J=0)-(J=2)):G
```

```
OTO 1030
1080 FOR J=0 TO 20
1082 P(2+2*INT(RND(1)*((NX-3)/2
)),1+INT(RND(1)*(NY-3)))=5
1085 P(1+INT(RND(1)*(NX-3)),2+2
*INT(RND(1)*((NY-3)/2)))=5
1088 NEXT
1090 SCREEN 2,0
1092 COLOR 1,10,4
1094 SPRITE$(1)= A$
1098 SPRITE$(0)= B$
1099 SPRITE$(2)=C$
1100 FOR J=2 TO NX-2
1105 FOR K=2 TO NY-2
1110 IF P(J,K)=5 THEN LINE (J*B
S+SX,K*BS+SY)-((J+1)*BS+SX-1,(K
+1)*BS+SY-1),4,BF
1130 NEXT K
1140 NEXT J
```

As linhas 1000 a 1080 "desenham" o labirinto na memória do computador e armazenam sua forma no conjunto P (cada elemento do conjunto corresponde a um bloco do labirinto). A subrotina armazena o número 5 em P onde quer que exista um caminho, e o número 0 faz o mesmo, caso exista uma parede.

Uma vez que o labirinto tenha sido armazenado em P, a linha 1090 liga a tela de alta resolução, e a linha 1092 define as cores de frente e de fundo do desenho. As linhas 1094 a 1099 definem os sprites correspondentes aos blocos gráficos montados na seção anterior do programa. Note que o sprite do bloco em branco tem prioridade sobre o bloco do homenzinho. As linhas 1100 a 1140 exibem o labirinto na tela mediante a verificação dos conteúdos de P. Quando um 5 for encontrado, um quadrado será impresso (comando **LINE** de bloco cheio) na linha 1110.

MONTE O JOGO

Agora você precisa de um jogo para divertir-se com o labirinto aleatório. Digite a próxima seção do programa; mas não o rode, pois ele chamará uma subrotina que ainda não existe.

```
1200 X=2:Y=2:LX=2:LY=2
1210 TI=800:LI=3
1215 R=RND(-TIME)
1220 TIME=0
1230 X1=1+INT(RND(1)*(NX-3))
1240 Y1=1+INT(RND(1)*(NY-3))
1250 IF P(X1,Y1)=5 THEN P(X1,Y1
)=7:PUT SPRITE 2,(SX+X1*BS,SY+Y
1*BS),10 ELSE 1230
1255 X1=X*BS+SX:Y1=Y*BS+SY
1260 PUT SPRITE 1,(X1,Y1),1
1270 T$=INKEY$:IF T$="" THEN 12
70
1280 IF ASC(T$)=28 THEN X=X+1
1290 IF ASC(T$)=29 THEN X=X-1
1300 IF ASC(T$)=30 THEN Y=Y-1
1310 IF ASC(T$)=31 THEN Y=Y+1
1320 IF P(X,Y)=7 THEN F=1:P(X,Y
)=5:GOTO 1330
1325 IF P(X,Y)<>5 THEN X=LX:Y=L
Y:GOTO 1270
1330 IF X<>LX OR Y<>LY THEN LX=
X:LY=Y
1340 IF F=1 THEN F=0:SC=SC+(TI-
TIME):TI=TI-10:GOTO 1220
1350 IF TIME>TI THEN GOSUB 1500
:IF LI<1 THEN 1000
1360 GOTO 1255
```

A linha 1200 contém as variáveis a partir das quais a posição do homenzinho é calculada. X e Y são as coordenadas da posição atual do homenzinho, e LX e LY, da última posição. **TI** é o limite de tempo para descobrir o tesouro, e LI, o número de "vidas" de que o jogador dispõe. O tempo limite é de cerca de dezesseis segundos; se este se esgotar sem que o tesouro tenha sido encontrado, o homenzinho perderá uma "vida". No início, o jogador tem três "vidas".

A linha 1215 introduz o gerador de números aleatórios, usando como "semente" o valor armazenado no cronômetro interno, naquele instante. O cronômetro interno do computador é zerado na linha 1220, antes que as linhas 1230 e 1240 selecionem uma posição aleatória para o tesouro. O elemento correspondente em P é examinado na linha 1250, para verificar se nesse lugar há um caminho ou uma parede. Se houver um caminho, o valor do conjunto será modificado de 5 para 7. A última parte da linha desenha o tesouro no labirinto.

A posição do pirata é calculada na linha 1255, levando-se em conta a largura do caminho, que é o tamanho do bloco BS. A linha 1260 coloca o homenzinho na tela, naquela posição.

As linhas 1270 a 1310 examinam o teclado através do comando **INKEY$**, de modo a permitir a movimentação contínua do homenzinho, no labirinto, comandado pelas teclas de controle do cursor. A linha 1325 faz com que ele se mantenha dentro do caminho, de modo a impedi-lo de atravessar alguma parede. A linha 1320 checa se o tesouro foi encontrado, examinando o elemento correspondente em P para o número 7. Quando o tesouro é encontrado, o "indicador de descoberta" (F) é igualado a 1.

A linha 1330 promove a movimentação do homenzinho, fazendo com que a localização atual torne-se a última posição. Como o homenzinho é definido por intermédio de um sprite, não é necessário apagá-lo do lugar onde estava, como em outros computadores.

A linha 1340 calculará o escore, se o tesouro for encontrado. O limite de tempo é diminuído de 10 segundos. O F retorna ao zero e o programa volta à linha 1220 para zerar o cronômetro. O programa continua, redesenhando o tesouro em outro lugar, mas deixando o pirata no ponto em que ele estava quando o último tesouro foi encontrado.

Se o jogador demorar muito para achar o tesouro, a linha 1350 chamará a sub-rotina que se inicia na linha 1500 (ela será apresentada a seguir). Essa sub-rotina diminui uma "vida" do jogador; caso tenha se esgotado o número de "existências", o programa retornará ao começo (linha 1000). Se o homenzinho não tiver achado o tesouro e ainda dispuser de algum tempo, a linha 1360 fará o programa voltar à linha 1255, permitindo que sua nova posição seja calculada.

EXIBA O ESCORE

A sub-rotina final exibirá o escore e o número de "vidas" restantes depois que uma "vida" for perdida:

```
1500 LI=LI-1
1505 PUT SPRITE 0,(X1,Y1),1
1507 OPEN "GRP:" FOR OUTPUT AS
#1
1510 PSET(0,0),10:PRINT #1,TAB(
3);"VIDAS:";LI;TAB(5);"SCORE =
";SC
1520 FOR J=1 TO 6000:NEXT
1525 LINE(0,0)-(255,7),10,BF
1530 IF LI>0 THEN PUT SPRITE 0,
(-20,-20):CLOSE #1:TIME=0:RETUR
N
1540 PSET(0,0),4:PRINT #1," Que
r jogar novamente ? (S/N)"
1550 A$=INKEY$:IF A$<>"S" AND A
$<>"N" THEN 1550
1560 IF A$="S" THEN RUN
1570 END
```

Rode o programa: se uma "vida" for perdida, ele diminuirá de 1 o número de "existências" restantes (linha 1500). A linha 1505 assinala a "morte" do personagem.

As linhas 1510 a 1530 exibem o nível de dificuldade, o número de "vidas" restantes e a contagem de pontos. Caso ainda sobrem "vidas", a linha 1530 colocará o cronômetro em 0.

As linhas 1540 a 1560 perguntam se o jogador quer tentar de novo; dependendo da resposta, elas param o programa ou o rodam novamente. O **RUN** é utilizado na linha 1560, limpando P para um novo labirinto.

UMA TÉCNICA DE ANIMAÇÃO DE BLOCOS GRÁFICOS NOS MICROCOMPUTADORES DA LINHA TRS-80.

Os micros da linha TRS-80, por serem de tecnologia mais antiga do que os da nova geração de computadores pessoais (Sinclair Spectrum, MSX, TRS-Color), contam com menores recursos para a definição de blocos gráficos (UDG, ou user = defined graphics*) além de terem uma resolução gráfica na tela (128 x 48 pixels) insuficiente para a programação de efeitos visuais mais sofisticados.*

Existem, porém, formas de contornar esse problema. Assim, para definir e animar blocos gráficos no TRS-80, utilizamos três conceitos básicos de programação: como incorporar caracteres gráficos a variáveis alfanuméricas (cordões); como usar os caracteres de controle do cursor para definir figuras complexas, e como movimentar blocos gráficos por meio do comando **PRINT@**.

Como a utilização desse comando já foi explicada, abordaremos aqui apenas a técnica de definição de figuras complexas.

Imagine que queremos definir um disco voador. Para montar a figura em uma única variável alfanumérica (por exemplo, **D$***) concatenamos os caracteres gráficos que determinam a primeira linha (veja no manual do seu computador os códigos gráficos correspondentes):*

```
A$ = CHR$(128)+CHR$(186)+
     CHR$(176)+CHR$(186)+
     CHR$(144)
```

Acrescentamos a seguir a linha do meio da figura. Mas, se nos limitarmos a concatenar os próximos caracteres gráficos ao **D$** *já definido, na hora de colocar o disco voador na tela por meio de um* **PRINT@**, *o desenho sairá errado (em vez de se posicionarem uma embaixo da outra, as duas linhas ficarão fundidas em uma única linha). Para evitar isso, temos que fazer com que o cursor se posicione no ponto certo, abaixo da primeira linha do gráfico. Para isso, usamos os códigos de controle:* **CHR$(26)**, *que tem o efeito de descer o cursor do* **PRINT** *de uma posição na tela, e* **CHR$(24)**, *que recua o cursor, sem limpeza.*

Dessa forma, para posicionar o cursor corretamente na linha seguinte do gráfico, temos que descer uma posição com o cursor e recuar cinco posições. Para economizar tempo de digitação, podemos definir um cordão **B$**, *que incorpora todos esses* **CHR$**:

```
B$ = CHR$(26)+STRING$(5,24)
```

E agora **A$** *fica definido assim:*

```
A$ = CHR$(128)+CHR$(186)+
     CHR$(176)+CHR$(186)+
     CHR$(144)+B$+CHR$(170)+
     STRING$(4,171)
```

Finalmente, podemos completar a figura, adicionando a terceira linha de blocos gráficos, e repetindo o truque de descer uma posição e recuar o cursor cinco posições:

```
A$ = CHR$(128)+CHR$(186)+
     CHR$(176)+CHR$(186)+
     CHR$(144)+B$+CHR$(170)+
     STRING$(4,171)+B$+
     CHR$(128)+STRING$(3,131)+
     CHR$(129)
```

O bloco gráfico está definido. Para desenhar, de uma vez só, esse bloco em um ponto qualquer da tela (por exemplo, na posição N), basta dar o comando:

```
PRINT@N,A$;
```

Para animar o bloco gráfico contido em **A$**, *variamos o valor de N, colocando-o dentro de um laço de programa. Se o valor de N for aumentado ou diminuído de 64, em cada repetição do laço, a movimentação se dará no sentido vertical. Por outro lado, se esse valor for aumentado ou diminuído de 1, o movimento se fará no sentido horizontal.*

Para apagar o bloco gráfico do lugar onde estava, é necessário colocar por cima dele um terceiro bloco gráfico previamente definido, contendo apenas três linhas de cinco caracteres em branco, concatenadas por meio do cordão de controle, **B$**, *definido acima.*

CONCURSO INPUT

2º Sorteio

5 microcomputadores para os sorteados do 1º ao 5º prêmios da Loteria Federal

REGULAMENTO

Para concorrer no 2º sorteio do sensacional Concurso INPUT, você deve juntar 06 selos que virão no canto inferior direito dos fascículos 1 a 16 (1 selo por edição), indistintamente. Se você enviou a cartela para o 1º sorteio, ainda sobraram 02 selos que poderão ser usados agora. Caso prefira não usá-los, envie os selos das demais edições citadas.

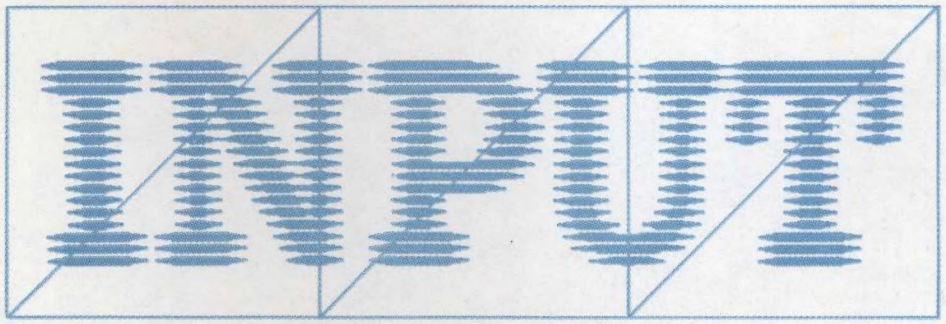
Cole os 06 selos na cartela abaixo, ou envie-os colados num papel dentro de um envelope.

Você receberá um número com o qual concorrerá no sorteio da Loteria Federal do dia 04 de outubro de 1986. Serão aceitas as cartelas/selos enviados até 05.09.86.

A cartela/selos deverão ser enviados juntamente com seu nome, endereço, telefone, nº da carteira de identidade (do participante ou responsável) e CPF, num envelope endereçado ao CONCURSO INPUT - Caixa Postal 9442, CEP 01000, São Paulo - SP.

OBS.: A prescrição do direito aos prêmios se dará 180 dias após o sorteio. O resultado será publicado nos fascículos INPUT e os prêmios entregues por nossos distribuidores nas cidades de residência dos contemplados.

Se tiver alguma dúvida, consulte nosso Serviço de Atendimento ao Leitor, pelo telefone (011) 815-8055 - ramal 235.



Nome:

Endereço:

CEP: Cidade:

Estado: Fone: Cédula de Identidade:

CPF:

Cert. Aut. Min. Faz. nº 01/00/237/86

Não perca mais esta chance de ganhar um dos 10 sensacionais microcomputadores HOT BIT HB 8.000 SHARP.
Continue colecionando INPUT. Seu micro ganha vida.

NOVA CULTURAL

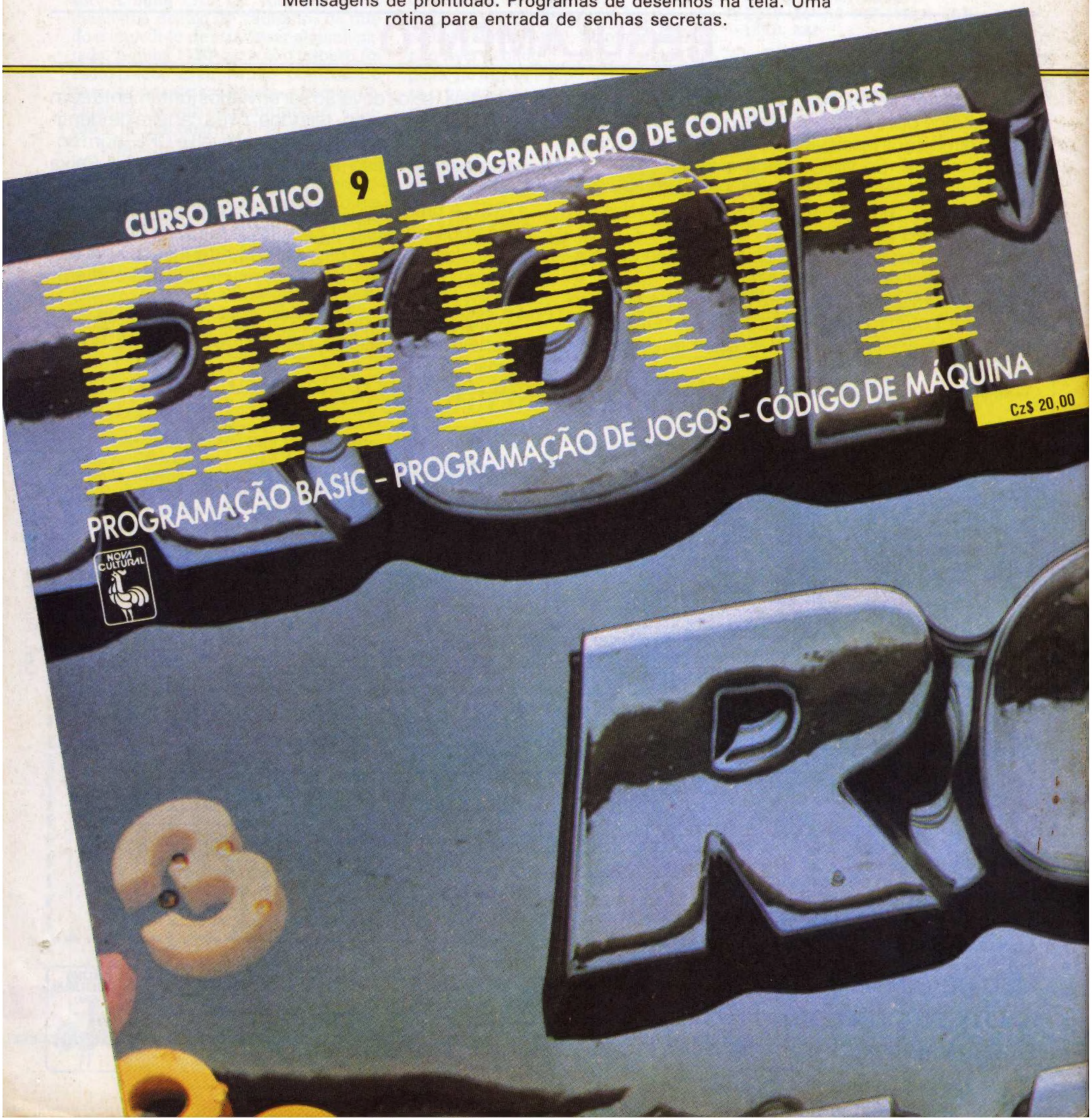
NO PRÓXIMO NÚMERO

PROGRAMAÇÃO DE JOGOS

Efeitos sonoros: das explosões aos ruídos extraterrenos e ao barulho de um trem, eles dão vida e "colorido" aos jogos.

CÓDIGO DE MÁQUINA

Como funciona a memória interna do computador. Memórias RAM e ROM. Onde os programas são armazenados.

PROGRAMAÇÃO BASIC

Mensagens de prontidão. Programas de desenhos na tela. Uma rotina para entrada de senhas secretas.